Algoritmo genético para el problema de la ubicación del controlador en una red definida por software (RDS)

(Tesis)

YEISON MORENO GÓMEZ

Director:

Felipe Díaz-Sánchez, Ph.D.

Codirector:

Juliana Arévalo, M.Sc.

UNIVERSIDAD SANTO TOMÁS

FACULTAD DE INGENIERÍA DE TELECOMUNICACIONES

MAESTRÍA EN TELECOMUNICACIONES Y REGULACIÓN TIC

BOGOTÁ, 2020

Dedicado a mis padres Inés y Luis quienes siempre pacientemente han estado a mi lado en todo momento brindando su apoyo y energía, a ellos les debo mi vida. Mis logros son sus logros, mis sonrisas son para ustedes, quien se siente orgulloso soy yo, porque el amor proporcionado con cada palabra y acto me hacen entender el sentimiento verdadero y puro de sus almas. Ustedes son mi universo y me encuentro completamente feliz de que sean mi familia.

Agradezco también a aquellos que algún día estuvieron y ahora no, abuelos los llevo en mis pensamientos.

Einige Menschen kommen in Form eines Lächelns und mit großartigen Lehren in dein Leben (TH).

AGRADECIMIENTOS

Agradezco al ser creador por brindarme una familia y personas tan importantes que rodean mi vida.

A mi director Felipe Díaz y codirectora Juliana Arévalo quienes con sus conocimientos y ayuda fueron parte del desarrollo y conclusión de este trabajo.

TABLA DE CONTENIDO

Lista de figuras

Lista de tablas

ACRÓNIMOS

API: Interfaz de Programación de Aplicaciones. Del inglés, Application Programming Interface.

CCPP: Problema de Ubicación del Controlador Capacitado. Del inglés, Capacitated Controller Placement Problem.

CNPA: Algoritmo de partición de Red Basado en Clústeres. Del inglés, Clustering-Based Network Partition Algorithm.

CO: Controlador. Del inglés, Controller.

CoH: Colonia de Hormigas . Del inglés, Ant Colony.

CPU: Unidad Central de Procesamiento. Del inglés, Central Processing Unit.

EA: Algoritmos de Evolución. Del inglés, Evolutionary Algorithms

FIFO: Primero en entrar, primero en salir. Del inglés, First-In, First-Out.

ITaaS: Infraestructura como servicio. Del inglés, IT as a Service.

LAN: Red de área local, Del inglés, Local Area Network.

LARC: Ubicación Confiable de Controladores con Reconocimiento de Latencia. Del inglés, Latency-Aware Reliable Controller Placement.

LCFM:Latencia de Configuración de Flujo Máximo.

LiDy: Algoritmo que ubica al controlador gracias a métricas de flujo dinámico. Del inglés, Algorithm that locates the controller thanks to dynamic flow metrics.

MLEC: Máxima latencia entre controladores.

MOCO: Problema de optimización combinatoria multiobjetivo. Del inglés, Multiobjective Combinatorial Optimization Problem.

MOGA: Algoritmo Genético Multi-Objetivo. Del inglés, Multi-Objective Genetic Algorithm.

NBI: Interfaz limitada hacia norte. Del inglés, NorthBound Interface.

NFV: Virtualización de funciones de red. Del inglés, Network Functions Virtualization.

NP-Hard: NP- Complejo. Del inglés, non-deterministic polynomial-time hardness.

NSFNET: Del inglés, National Science Foundation's Network.

NSGA-II: Algoritmo genético de clasificación no dominado II. Del inglés, Non-dominated Sorting Genetic Algorithm II.

OEP: Optimización de enjambre de Partículas.

OS3E: El intercambio abierto de ciencia, becas y servicios. Del inglés, The Open Science, Scholarship and Services Exchange.

PN: Porcentaje de nodo.

POCO: Óptima ubicación del controlador basada en Pareto. Del inglés, Pareto-based Optimal COntroller-placement.

PUC: Problema de Ubicación del Controlador. Del inglés, Controller Placement Problem.

RAM: Memoria de Acceso Aleatorio. Del inglés, Random Access Memory.

RDS: Redes Definidas por Software.

SBI: Interfaz limitada hacia el sur. Del inglés, SouthBound Interface.

SC: Agrupación espectral. Del inglés, Spectral Clustering.

SEF: Solicitud de establecimiento de Flujo.

SW: Dispositivo para reenvío de información. Del inglés, Forwarding Device, Switch, node.

TM: Tasa de mutación.

WAN: Red de área amplia. Del inglés, Wide Area Network.

RESUMEN

Las redes tradicionales están siendo transformadas gracias a las redes definidas por software (RDS) debido a sus características de flexibilidad, fácil administración y cambios rápidos sobre la red, estas modificaciones son un paradigma de la industria de las redes y posibles gracias a la programación. Este trabajo se centra en proponer un resultado al Problema de la ubicación del controlador (PUC) a través de un algoritmo genético, en el cual se consideran los movimientos de controladores y enlaces, para esto se consideraron los datos de algunas topologías del “Topology Zoo” y se respetaron las restricciones tenidas en cuenta en el modelo, para implementar el modelo propuesto se utilizó Python.

Palabras clave: RDS, PUC, algoritmo genético, Topology Zoo, movimientos de controladores y enlaces.

ABSTRACT

Traditional networks are being transformed thanks to Software Defined networks (SDN) due to their characteristics of flexibility, easy administration and fast changes on the network, these modifications are a paradigm of the network industry and possible thanks to programming. This work focuses on proposing a result to the CPP through a genetic algorithm, in which the movements of controllers and links are considered, for this the data of some topologies of the "Topology Zoo" were considered ad been respected in account in the model, to implement the proposed model Python was used.

Keywords: SDN, CPP - Controller Placement Problem, Genetic Algorithm, Topology Zoo, controller and links movement.

INTRODUCCIÓN

Las redes son fundamentales en cualquier entorno, ya sea empresarial, social o en el ámbito de los negocios. Las redes actualmente se pueden encontrar en lugares físicos, en la nube o en ambos escenarios y satisfacen una de las necesidades básicas de la humanidad que es la comunicación, gracias a las redes se pueden prestar servicios y crear ambientes de competencia. RDS representa la evolución de las redes tradicionales [1], puesto que su configuración y control hacen que sea más ágil y sencillo administrar las redes.

RDS permite virtualización, lo cual significa garantizar que los recursos son utilizados de una manera más apropiada y que la red más fácil utilizar, también hace que los procesos de implementación manual de las redes tradicionales sea dinámico, rápido y efectivo [2]. Esta virtualización también permite un aumento considerable de las capacidades sin que el costo económico sea alto.

Dentro de los beneficios de RDS se encuentra la manera de aprovisionar servicios de manera automática, al igual que la configuración y la capacidad de realizar correcciones sobre la red en caso de fallos, mientras que en las redes tradicionales se tiene que configurar cada hardware uno a uno, RDS permite que los cambios se realicen a través de actualizaciones por software.

Estos beneficios son mucho más económicos que los tradicionales, se suele pensar que lo nuevo e innovador es costoso, pero en el caso de RDS, el retorno de inversión (ROI) es rápido debido a sus beneficios casi instantáneos, ya que la red se convierte en resiliente y dinámica.

Otro beneficio de RDS son las políticas para la administración de la red que se convierten en simples y son orientadas a la aplicación, lo cual hace que sea mucho más sencillo manejar la red, estas políticas se ajustan a la automatización y crean una herramienta de alto nivel de fácil manejo.

Las RDS son posibles gracias a los controladores [3], los cuales gobiernan los switches y los diversos flujos de información que circula a través de los dispositivos que maneja el controlador, gracias al controlador la red puede ser administrada en términos de balanceo de carga, tolerancia de fallos, resiliencia [4], latencia [5], escalabilidad, rendimiento.

Por lo tanto uno de los más grandes retos en RDS es saber ¿dónde ubicar el controlador dentro de la red? y determinar ¿cuántos controladores deben ubicarse dentro de la red? [6], para encontrar la respuesta, se han realizado trabajos que atacan este problema desde métodos exactos y métodos heurísticos, ya que el controlador es un recurso finito y su despliegue involucra un costo adicional, incrementando el Capex y Opex [7].

En este trabajo se propone una respuesta a la ubicación del controlador y la cantidad de controladores que se usarán, teniendo en cuenta el costo a optimizar y el tamaño de la red, se implementa un algoritmo genético que hace parte de la familia de algoritmos heurísticos.

1. MARCO GENERAL DEL PROYECTO

1.1 OBJETIVOS

Objetivo General

Proponer y evaluar un algoritmo genético para el problema de ubicación del controlador en redes WAN RDS.

Objetivos específicos

- Identificar los trabajos relacionados de métodos heurísticos sobre la ubicación de los controladores en RDS WAN para establecer los criterios de optimización, las restricciones, las técnicas y las topologías utilizadas para tratar este problema.

- Proponer un algoritmo genético para abordar el problema de ubicación del controlador teniendo en cuenta factores como los criterios de optimización, las restricciones, las técnicas y las topologías.

- Implementar el algoritmo genético para abordar el problema de ubicación del controlador en redes WAN RDS.

- Validar el algoritmo propuesto midiendo su desempeño con respecto a los criterios de optimización, restricciones y topologías.

1.2 ALCANCE

El alcance de la propuesta incluye el desarrollo de un algoritmo genético que identifica la mejor ubicación para el controlador RDS en las redes WAN, considerando los criterios de optimización, las restricciones y las topologías identificadas en la revisión (objetivo específico 1).

El algoritmo genético se implementará en un software de computación numérica (por ejemplo, Python). La implementación se validará con redes basadas en el "*topology zoo*" (zoológico de topologías). La propuesta no incluye el uso de simuladores de red (emuladores) para modelar, implementar o validar el algoritmo genético.

1.3 METODOLOGÍA

El proyecto tiene 4 paquetes de trabajo como se observa en la figura 1.1:

- **Identificación de criterios para el modelamiento del problema.**

Se realizará revisión de los trabajos relacionados con el PUC. Para ello, se realizará una búsqueda en las bases de datos que ofrece la Universidad. Esta revisión permitirá identificar objetivos de optimización, restricciones, técnicas utilizadas y topologías evaluadas en la literatura científica. Este paquete de trabajo también incluye la identificación de las herramientas de software y hardware utilizadas para solucionar el problema.

- **Generación de modelos.**

A partir de la información recopilada en la fase de generación de insumos, se identificarán los métodos propuestos e implementados por otros autores para seleccionar los más apropiados de acuerdo con los parámetros a optimizar. Este paquete de trabajo también incluye la descripción de modelos y la definición de escenarios para la evaluación de modelos.

- **Evaluación del modelo.**

El modelo propuesto se simulará en un lenguaje de programación de alto nivel y las simulaciones se llevarán a cabo hasta que se logre una generalización de los resultados. Este paquete de trabajo incluye la implementación, validación y evaluación del desempeño del modelo propuesto. Si es necesario, se realizarán múltiples iteraciones entre este y el segundo paquete de trabajo.

- **Disertación**

En este paquete de trabajo se elabora la tesis. La disertación presentará una descripción del modelo, los resultados y los hallazgos principales.

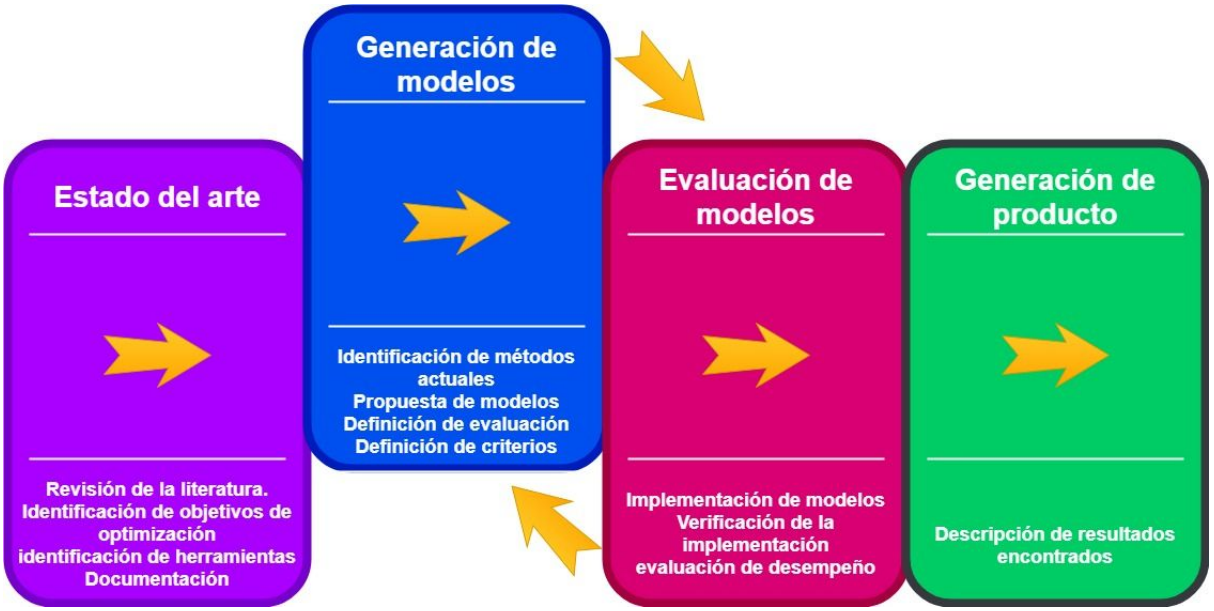


Figura 1.1. Metodología del proyecto.[Gráfica de elaboración propia]

2. Redes Definidas por Software y el problema de ubicación del controlador

Esta sección proporciona una explicación sobre las redes definidas por software. (RDS [1]), topologías donde las RDS han sido desarrolladas, la definición de controlador [8] y funciones [9], el Problema de Ubicación del Controlador (PUC) [6], y finalmente se dará una breve descripción de los métodos heurísticos [10] enfocados en el método genético [11].

2.1 Visión general de RDS

Internet es una supernet donde casi todas las personas están conectadas desde cualquier parte del mundo. Aunque esta tecnología es bien recibida, las redes tradicionales son complejas e inflexibles [12], son difíciles de configurar debido a las políticas estáticas y vulnerables a cambios inesperados, fallas y sobrecargas.

Las redes tradicionales tienen problemas significativos, como una transición de IPv4 a IPv6 [13], Eso comenzó hace algunos años y continúa sin terminar debido a su complejidad [14].

La mayoría de las redes que están implementadas actualmente tienen los planos de datos y de de control que se encuentran agrupados. Las redes definidas por software (RDS) son un paradigma [15] que ofrece cambiar el concepto tradicional de red, mediante la separación del plano de control del plano de datos de los diferentes dispositivos (enrutadores, conmutadores, dispositivos de reenvío). El modelo de planos independientes conduce a una lógica centralizada que controla la red y, por lo tanto, la programación y gestión de la red [16].

Las redes actuales están integradas verticalmente, es decir, los planos de control y de datos están vinculados [17]. El plano de control decide cómo se distribuye el tráfico en la red. Mientras que el plano de datos reenvía los paquetes de acuerdo con las decisiones tomadas por el plano de control. Esta arquitectura ha sido muy exitosa, ya que es altamente resistente. En particular, si un nodo falla, el tráfico en la red se puede distribuir por otras rutas, ya que todos los nodos integran las funciones de enrutamiento y reenvío de paquetes. Sin embargo, el aumento en el tamaño de estas

redes las hace mucho más complejas en aspectos como el diseño y la administración [18].

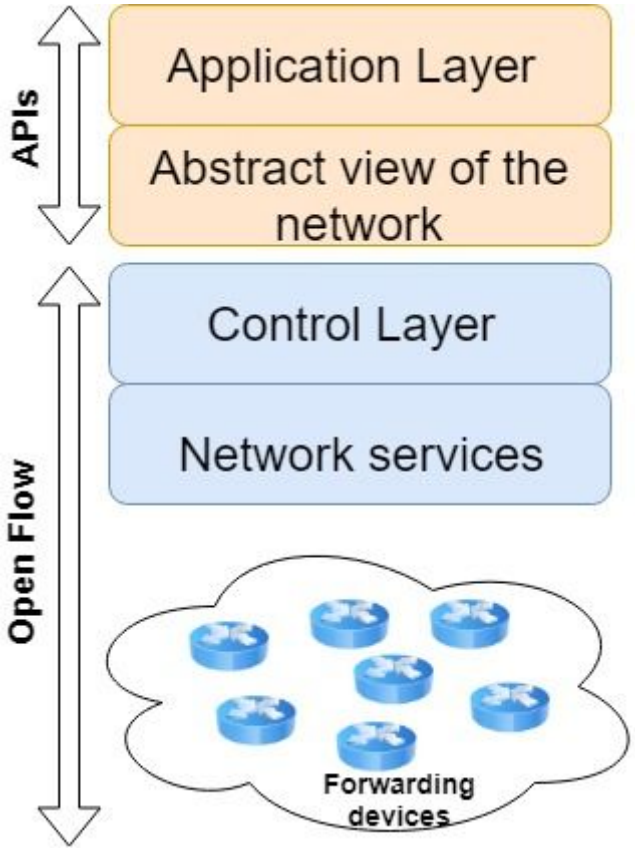


Figura 2.1. Arquitectura básica de RDS.[Gráfica de elaboración propia]

Las RDS se han convertido en un paradigma tecnológico que simplifica la gestión de la red y permite el desarrollo de innovaciones basadas en la separación de los planos de control y de datos. Una RDS se compone de tres capas como se observa en la figura 2.1: i) la capa de aplicación, que es una de las capas RDS más características, ya que comunica los requisitos de red y el comportamiento deseado de la red al controlador RDS mediante la interfaz NorthBound (NBI) [19]. Es decir, a través del software, esta capa implementa funciones de red tales como: enrutamiento, cortafuegos, balanceo de carga, entre otros. ii) La capa de control, responsable de administrar los dispositivos de reenvío (conmutadores) a través del controlador. Con esta capa, se crea una vista centralizada de la red, para administrar el flujo de datos en la capa de infraestructura (capa de datos) que está conectada a través del protocolo OpenFlow. iii) Finalmente, la capa de datos es responsable de reenviar los paquetes según la configuración de

los dispositivos, que previamente son manipulados por el controlador [20]. La implementación más simple de una RDS consiste en un controlador lógicamente centralizado, que tiene una visión global de la red.

Esta separación de planos implementada en el hardware (enrutadores, conmutadores, dispositivos de reenvío) genera la flexibilidad requerida, ya que minimiza el esfuerzo de gestión de la red y facilita su evolución.

El plano de control y el plano de datos se separan en redes RDS. El OpenFlow [21]. El protocolo actúa como las interfaces y es el responsable de la comunicación entre el plano de control y el plano de datos [22]. Como se observa en la figura 2.1 en la arquitectura básica de RDS.

RDS se puede describir en cuatro características:

- El plano de control y el plano de datos están desacoplados [23]. El control de los dispositivos (enrutadores, conmutadores, dispositivos de reenvío) se elimina, estos elementos se convierten en dispositivos para el envío de paquetes.
- En las redes tradicionales, el reenvío de paquetes se basa en el destino. En RDS el reenvío de paquetes se basa en la tabla de flujo [7].
- El controlador RDS es un elemento nuevo en la red, es una unidad de control centralizada y lógica. Es un software que ayuda a programar los dispositivos de reenvío [24].
- Las aplicaciones de software que se encuentran en el nivel más alto del sistema operativo de red implícito con los dispositivos del plano de datos [25]. Esta es una de las características más importantes en RDS.

2.2. El Controlador

El controlador en RDS es una aplicación en una arquitectura de red definida por software que gestiona el control de flujo para mejorar la gestión de la red y el rendimiento de la aplicación [26].

El controlador en RDS tiene la base de datos centralizada de las políticas de control de la red y la infraestructura de las aplicaciones [3]. Su objetivo es proporcionar servicios

de red en redes virtuales, cableadas, inalámbricas, para LAN y WAN. Cuando se desee, se pueden implementar diferentes políticas en el controlador a través de las APIs.

El controlador es el que reúne los dispositivos de red y las aplicaciones, de esta forma la configuración y administración de la infraestructura se automatizan. El controlador tiene una interfaz programable que se conoce como dirección norte [27]. El controlador es el que reúne los dispositivos de red y las aplicaciones, de esta forma la configuración y administración de la infraestructura se automatizan. El controlador tiene una interfaz programable que se conoce como dirección norte.

Para reducir la carga de la administración de la red, el controlador automatiza la implementación y las políticas de red en su entorno, otras funciones esenciales del controlador son [8]:

- Consistencia en la red, lo que genera una reducción en el tiempo de inactividad, la complejidad y los costos (monetarios).

- Configuración y aprovisionamiento automático de punta a punta que genera un despliegue más eficiente en servicios y aplicaciones.

- Admite implementaciones nuevas y existentes, gracias a la capacidad de programación y la infraestructura que tiene.

2.3. Problema de Ubicación del Controlador

La primera vez que el problema de ubicación del controlador (PUC) fue propuesto en [6] donde se exponen dos preguntas; la primera es: "¿Cuántos controladores se necesitan?" y la segunda pregunta es: ¿dónde deben estar ubicados para una topología dada? *Heller et al.* propone como las posiciones de los controladores deben determinarse con respecto a la latencia media y la mayor latencia desde los contralores a los switches (dispositivos de envío). En este caso, no se consideró el balanceo de carga de los controladores que proviene de las solicitudes de otros nodos.

Los autores en [14] consideraron los nodos con diferente peso e importancia y los flujos entre ellos también se consideran la capacidad del controlador (como una memoria y el procesamiento de la CPU). Por esta razón, se propuso un algoritmo para ubicar un controlador de manera estática, cuando este controlador está sobrecargado, otro algoritmo migrará el controlador.

En [28] *Guo et al.*, exponen la falla de la red como una variable para resolver el PUC, para este caso, el autor consideró las fallas de dos maneras reconocidas como estados, la primera como una falla de nodo y la segunda como una falla de enlace, este problema de ubicación del controlador modelado se considera una falla en un solo enlace.

En [22] se enfocó en el el método llamado AE (agrupación espectral), que divide la topología en pequeños dominios RDS (grupo de dispositivos de reenvío como switches) y comienza a partir de esto para encontrar la ubicación del controlador en cada dominio RDS. Sin embargo, este método tiene un problema, porque cuando la topología se divide, el algoritmo no examina los nodos más remotos, y esto significa que puede producir un nuevo dominio entre estos nodos remotos. En otras palabras, el algoritmo no encontrará la mejor ubicación del controlador.

Por lo general, los trabajos sobre PUC se aplican a topologías pequeñas y topologías medianas en redes RDS, porque para estos casos solo es suficiente un controlador para manejar la red. Sin embargo, cuando se revisa el problema de las redes de gran tamaño, hay más de una métrica a considerar y el número de controladores aumenta, todo esto estará vinculado al tipo de red e infraestructura disponible.

En las siguientes secciones, hay una revisión de la literatura, organizada en dos grupos, que hablan sobre la ubicación del controlador:
i) Método Estático, que realiza el cálculo de(l) el/los controlador(es) una sola vez.
ii) Método Dinámico, que recalcula la ubicación y cantidad del/los controlador(es) cada vez que hay una cambio en la red.

2.3.1. Principio de Pareto en redes estáticas

Uno de los métodos más utilizados para encontrar la solución del PUC es el principio de Pareto [4], que se utiliza para redes de pequeña y mediana escala. Teniendo en cuenta las fallas del controlador es esencial. Si el controlador central falla, los nodos deben buscar otro controlador, lo que significa que la red tendrá nuevas latencias asociadas con el nuevo controlador asignado. La pérdida del controlador central puede significar que algunos nodos no pueden conectarse a otro controlador, por lo que se deben tener en cuenta varias métricas para encontrar una ubicación adecuada para el nuevo controlador.

Se ha demostrado que no es suficiente considerar la métrica de latencia entre el nodo y el controlador, sino que también se deben tener en cuenta otras métricas. Este trabajo [29] muestra, además de la latencia entre el nodo y el controlador, otros parámetros como la resiliencia, el equilibrio de carga y la latencia entre controladores. Se utilizó Matlab, tomando el "*Topology Zoo*". El algoritmo utiliza el óptimo de Pareto y permite comparar las diferentes ubicaciones del controlador de acuerdo con las métricas tomadas en cuenta. El resultado fue la opción de seleccionar varias ubicaciones de controlador según un parámetro en particular.

Otros estudios realizados consideraron una o más métricas para determinar la posición del controlador. No hay una respuesta única para el PUC, sino más bien un conjunto de soluciones que implica una compensación. Dado esto, en [30] el algoritmo denominado como *Pareto Optimal Controller* (POCO), realiza una revisión exhaustiva de todas las ubicaciones posibles del controlador. Aunque este método es factible para redes pequeñas y medianas, no lo es para redes a gran escala o dinámicas, ya que requiere mucho tiempo para el cálculo y tiene limitaciones de recursos como el procesamiento y la memoria. En este enfoque, los autores utilizan POCO combinado con el método heurístico [31] de recocido simulado.

El método heurístico analiza la compensación entre el tiempo y la precisión. Toma las latencias de los nodos, las latencias entre los controladores, la tolerancia a fallos en los nodos, la tolerancia a fallos en los enlaces y el equilibrio de carga. Se utilizó el "*Topology Zoo*" para evaluar el rendimiento de esta propuesta. Los resultados

mostraron que este método es menos preciso pero más rápido para determinar la mejor ubicación del controlador.

Si bien la mayor parte de la documentación de RDS dedicada a la ubicación del controlador, presenta el problema de la posición como "NP-Hard", las soluciones a este problema muestran altos tiempos de procesamiento de la CPU, gasto de memoria y tiempo de procesamiento. En [4] *David Hock et al.* propone un algoritmo para la ubicación de controlador basada en Pareto (POCO) de una manera resiliente (con capacidad de recuperación de fallos), que evalúe todas las soluciones posibles a la ubicación del controlador teniendo en cuenta las métricas de resistencia y tolerancia a fallas. Este método fue evaluado en una topología OS3E internet2; Los resultados revelan una mejora en los tiempos de cálculo, un uso más eficiente de la memoria y la RAM.

Los estudios más recientes sobre el PUC y Pareto, buscan que el algoritmo sea dinámico, en este enfoque, los autores describieron el método POCO-PLC [32] aplicado de manera convincente al PUC en redes RDS. Utilizando la red de investigación "Planet Lab" programada en Python que mide la latencia entre nodos externos y la latencia entre nodos locales en tiempo real, así como la carga en la CPU. Luego, "Planet Lab" elige dinámicamente el controlador más cercano al nodo con respecto a la latencia. Dado que la asignación del controlador es dinámica, permite comparar en tiempo real las diferentes ubicaciones del controlador para las ubicaciones anteriores.

La ubicación del controlador se ha convertido en una tarea crítica en las redes RDS. Al tratar de ubicar varios controladores, cada área afecta las métricas, creando un conflicto. Por ejemplo, considerando la latencia entre los controladores, si se encuentran cerca uno del otro, la métrica de latencia del controlador a un nodo se ve afectada. Este problema se conoce como "Problema de optimización combinatoria multiobjetivo (MOCO)", que se ha trabajado utilizando la "frontera de Pareto" [33]. Este método es útil en problemas combinatorios pequeños y medianos, al encontrar la ubicación del controlador en un tiempo razonable, pero en problemas a gran escala, este algoritmo involucra miles de millones de soluciones, que se traducen en memoria consumida y altos tiempos de procesamiento.

Para este caso se usó la segunda versión del NSGA-II (Algoritmo genético de clasificación no dominado) mezclado con la frontera de Pareto. Este algoritmo genético generalmente se usa en problemas de optimización discretos y continuos [34], y este modelo se lleva al problema de ubicación del controlador [35]. Se realizó una simulación de Matlab de las topologías "Internet2 y OS3E", el algoritmo NSGA-II encontró un enfoque diverso y preciso de optimización de Pareto y tiene en cuenta la latencia entre los controladores, la latencia entre el controlador y el nodo, la latencia entre los nodos y el equilibrio de carga. Este algoritmo “NSGA-II” fue eficiente para encontrar las aproximaciones "Óptimos de Pareto". Esta investigación muestra las ventajas de los métodos de evaluación exhaustivos que se encargan de encontrar una solución en todo el espacio.

2.3.2. Algoritmo codicioso en redes estáticas

Un algoritmo "codicioso" de elección de ubicación aleatoria. Este algoritmo ubica los controladores en la red de manera iterativa en diferentes lugares

Con respecto a los algoritmos “codiciosos” [36] utilizados en el PUC, hay pocos intentos, en una RDS. Las estrategias de ubicación del controlador influyen en la red, desde el nodo hasta las latencias del controlador, hasta el rendimiento de la red.

Por esta razón, es esencial generar un algoritmo que proporcione una ubicación óptima del controlador. El objetivo de este artículo [37] es proporcionar un lugar óptimo de/los controlador(es) si se tiene en cuenta que existe una red con un número finito de controladores.

La confiabilidad consideró una sola métrica porque si los enlaces entre los dispositivos de reenvío y los controladores fallan, los planos de control y reenvío de datos se desacoplarán.

El algoritmo codicioso primero, hace una lista de las posibles posiciones y calcula el costo más bajo para cada ubicación posible, en la segunda iteración, toma un controlador con las rutas que convergen en él y toma la ruta más corta en el tiempo, este proceso lo repite con todos los controladores . El algoritmo evaluado en una

topología de la red Internet2 OS3E y las topologías de Rocketfuel. Los resultados muestran ubicaciones de controladores muy bien elegidas y confiables para la red.

Otros autores aceptan mezclar diferentes tipos de algoritmos, considerando la ubicación del controlador en las redes RDS, lo cual es un problema importante, ya que la demora y la sincronización de la red se verán afectadas. En [38] se busca mejorar la confiabilidad de la red, se considera la ruta más corta entre el controlador y los dispositivos de reenvío, un algoritmo de optimización global mezclado con un algoritmo de optimización codicioso es utilizado. Los resultados de la simulación evidencian un mejor rendimiento de los algoritmos propuestos en las topologías OS3E y *topology zoo*.

Otro trabajo relacionado con "codicioso", para encontrar la solución del PUC que aparece en [39] Md Tanvir [et al.] Propone elegir la ubicación del controlador para minimizar la latencia entre los dispositivos de reenvío y los controladores y determinar el número de controladores en la RDS .

Los autores consideraron la carga dinámica. Ellos crearon el algoritmo Lidy + (Conformado para O (n2): Algoritmo para descubrir pares de patrones booleanos óptimos y un algoritmo para flujo dinámico), una topología conectada a 600 usuarios que lo probaron.

Los resultados evidenciados son: baja cantidad de controladores en la red probados, poco costo de energía y mantenimiento respecto del algoritmo anterior Lidy.

2.3.3. Otros algoritmos en redes estáticas

Las primeras muestras de estudios realizados en el PUC se realizaron teniendo en cuenta una red estática. Es decir, la ubicación del controlador se determinó en un momento preciso, haciendo que la solución sea estática, teniendo como entrada principal la latencia entre los controladores y los dispositivos de reenvío.

La mayoría de las investigaciones sobre el PUC no contemplaron la confiabilidad de la comunicación o la latencia entre los controladores en el caso de la falla de un enlace.

En [5] *Yuqi Fan et al.* y otros, tomaron el tiempo de comunicación entre los dispositivos de reenvío y el controlador, teniendo en cuenta la falla de un solo enlace.

En este caso, se diseñó un algoritmo llamado "Ubicación de Controladores Confiables A Través de la Latencia (LARC)" para buscar la ruta más corta en el tiempo, entre el controlador y los dispositivos de reenvío. Se utilizaron las topologías AT&T (Norteamérica) e Internet 2.

Los resultados muestran una menor latencia en los enlaces principales y los enlaces de respaldo entre los controladores y los dispositivos de reenvío cuando falla el enlace que conecta estos dispositivos.

La versión más reciente del algoritmo NSGA elaborada por *Lingxia Liao et al.*, proponen en [40] un problema de ubicación de controlador distribuido que descubre el óptimo de Pareto para reducir el retraso entre los dispositivos de reenvío(switches) a controlador(es), y controlador(es) a controlador(es), también el balanceo de carga del controlador. Propusieron un Algoritmo Genético de Objetivos Múltiples (MOGA) que considera métricas de objetivos múltiples como el ahorro de energía, la migración del controlador o la asignación de NFV, este algoritmo simulado en Matlab con el repositorio "*Rocketfuel*" (Sistema Autónomo 3561). Los resultados mostraron la mejor diversidad en las posiciones globales otorgadas para la Frontera de Pareto, y se redujo el tiempo respecto al algoritmo anterior llamado MOGA (trabajo anterior).

Otro punto de vista en [27] define el problema de ubicación del controlador capacitado (PUCC)", considerando la latencia de propagación entre el controlador y los dispositivos de reenvío, y la carga de los controladores.

La carga de un controlador consta de cuatro partes [41]: primero, paquetes de entrada y entrega a las aplicaciones. En segundo lugar, la partición de la red local. Tercero: separación de la visión global. En cuarto lugar, la instalación de las entradas de flujo generadas por las aplicaciones. Cuando muchos paquetes de entrada llegan al controlador, el rendimiento y el ancho de banda disminuyen.

El objetivo de [27] es considerar la carga del controlador; por lo tanto, se utiliza un algoritmo de propagación entera para encontrar el número mínimo de controladores, se utilizó "*Topology Zoo*" (82 de 261 topologías).

Los resultados muestran una reducción en el número de controladores; También se redujo la carga de los controladores activos.

Otros autores en [42] examinan en primer lugar los últimos métodos relacionados con PUC en RDS. En segundo lugar, propone un enfoque para reducir la latencia entre los controladores y los dispositivos de reenvío/switches.

El problema del controlador ha causado mucha investigación. Afecta directamente a las RDS, en métricas como el rendimiento, la tolerancia a fallos, la resistencia, la latencia y otros. El "Algoritmo de partición de red basado en agrupación (CNPA)" [42] es un algoritmo propuesto que buscó un nuevo enfoque para encontrar la mejor ubicación del controlador, que divide las redes en subredes y ubica los controladores en ellas, para reducir la latencia entre el controlador y el controlador a los dispositivos de reenvío.

El algoritmo fue simulado en Matlab, con una topología OS3E. Los resultados mostraron que la latencia de la red entre los controladores y los dispositivos de reenvío/switches, el costo de implementación y el consumo de energía se minimizaron, mientras que la confiabilidad y la capacidad de recuperación aumentaron al máximo.

Los estudios han demostrado que el PUC se puede atacar más fácilmente si se usan métodos heurísticos, en [43] [Erika Tarazona] et al., Elaboraron una encuesta de las técnicas más conocidas sobre la colocación del controlador en RDS.

Otro método heurístico propuesto como lo es "Controlador óptimo de Pareto", en el cual la ubicación de los controladores se usan en una red a gran escala en RDS.

Otro trabajo, de Stanislav Lange [44], propone un algoritmo heurístico para dar una compensación entre el tiempo de cálculo y la precisión del controlador.

Es necesario optimizar múltiples objetivos al mismo tiempo, por eso se utiliza el enfoque multicriterio. Sin embargo debe ser factible en el cálculo y la optimización. Por este motivo, se utilizan métodos heurísticos que permiten realizar soluciones precisas en relación con el tiempo en topologías a gran escala.

2.3.4. Algoritmos de tolerancia a fallos en redes dinámicas

Las redes RDS tienen el plano de control separado del plano de datos; esta división presenta problemas de confiabilidad, la brecha entre la capa de control y la capacidad de datos resulta en la pérdida de paquetes.

Ante este problema, dada una falla de cada componente de la red, por ejemplo, controladores, dispositivos de reenvío o enlaces. *Yannan Hu et al.* en [45] proponen determinar cuántos controladores y cómo conectarlos para optimizar la confiabilidad de la red.

Para esto, los autores introdujeron una sección llamada "Porcentaje esperado de pérdida de la ruta de control", lo cual es el número de rutas de interrupción del control debido a fallas de línea, en este punto se definen los dispositivos de reenvío del controlador y las rutas entre controladores, considerando la latencia entre ellos.

Se evaluó con varios algoritmos, como el recocido simulado en la topología OS3E. Los resultados muestran que las soluciones están cerca de ser óptimas y que la cantidad de controladores a ubicar debe ser correcta. También que muchos o pocos controladores, reducen la fiabilidad de la red.

Otro autor que trató la importancia en las redes definidas por software sobre la tolerancia a fallas es *Francisco J Ros et al.* en [43] quién dijo que se requiere más de un controlador y estos deben estar en diferentes ubicaciones. Centrado en determinar la ubicación, y cuántos controladores son necesarios para implementar y qué nodo está bajo el dominio del controlador, esto es posible considerando la dirección sur (southbound) (interfaz entre el nodo y el controlador). Se implementó un algoritmo heurístico en 124 topologías de red disponibles; los resultados mostraron la posibilidad de lograr la tolerancia a fallos en las redes RDS que determinan la ubicación de los controladores.

2.3.5. Otros algoritmos en redes dinámicas

La gran mayoría de los estudios realizados con respecto al PUC en redes RDS consideran valores estáticos de una red (es decir, las latencias entre nodos nunca cambian). El primer estudio que se consideró como aprovisionamiento dinámico para el PUC en RDS fue propuesto por [D. Hock] et al. [32].

En las redes RDS implementadas a gran escala, el PUC se basa solo en un momento del estado de la red. Tiene limitaciones como el rendimiento y la escalabilidad, por esta razón, es necesario ubicar más de un controlador en la red.

Sin embargo, existe un problema denominado "Problema de aprovisionamiento del controlador dinámico (PACD)", que consiste en adaptar dinámicamente el número de controladores y sus ubicaciones con los cambios constantes de la red. El PACD ajusta dinámicamente el número de controladores activos y los distribuye a varios dispositivos de reenvío.

Para realizar estos ajustes dinámicos, se utilizaron dos algoritmos heurísticos programados en Python. El primero fue el DCP-Gk que utiliza el problema de la mochila codiciosa y el DCP-SA que se basa en el recocido simulado. Se simuló en la topología RF-I (79 nodos, 294 enlaces) y RF-II (108 nodos, 306 enlaces).

Los resultados muestran la disminución de la sobrecarga de comunicación y el tiempo de configuración del flujo. La configuración de flujo es la instalación de una nueva ruta en el controlador propuesto por los dispositivos de reenvío (switches).

Los autores *Hemant Kumar Rath, et al*., Propusieron en [46] un algoritmo de colocación del controlador basado en los juegos de una teoría que involucra la heurística computacional. Este algoritmo fue implementado en Matlab y las latencias entre los dispositivos de reenvío y el controlador fueron consideradas, así como el equilibrio de carga en los controladores. Se simularon en una red aleatoria de 28 dispositivos de reenvío. Los resultados mostraron una carga homogénea en los controladores, el uso eficiente de los controladores a su capacidad máxima y la reducción de la latencia entre los dispositivos de reenvío y el controlador.

La asignación dinámica del controlador es muy conveniente ya que, en las redes RDS, los controladores deben estar conectados a los dispositivos de reenvío para evitar eventos dañinos en la red. Los problemas en la red pueden aislar los dispositivos de reenvío y la sobrecarga en la red.

Lucas Müller et al. proponen "survivor" [47], que refuerza la conectividad del controlador al reenvío de dispositivos, toma más rutas, evita la sobrecarga del controlador y mejora la conmutación por error al tomar listas de enrutamiento como copias de respaldo. Gracias a un algoritmo heurístico que evalúa la ubicación de los controladores actuales y determina si deben cambiar de posición o no, hacer comparaciones entre los controladores. Este método se evaluó con la topología de Internet 2, tomada de *Topology Zoo*, considerando (10 nodos, cinco enlaces), RNP (27 nodos, 15 enlaces) y Grant (40 nodos, 61 enlaces).

El flujo de datos juega un papel vital en la determinación de la ubicación del controlador. Los trabajos desarrollados en RDS muestran posibles escenarios que mejoran las redes de comunicación; sin embargo, la asignación de un controlador entre los switches contempla situaciones estáticas, lo que genera un desequilibrio de carga en condiciones de flujo dinámico. Long Yao [et al.] En [45] propone un algoritmo de migración de conmutador, que migra la carga de los conmutadores a otros controladores cuando se detectan ráfagas de flujo. Este algoritmo se probó en una topología NSFNET y consideró el peso de los conmutadores/switches y el retardo de conmutación al controlador; Los resultados mostraron una mejor distribución de la carga en momentos de congestión de la red.

El PUC considera el tráfico generalmente fijo, y actualmente no hay soluciones para encontrar la carga. En [42], se muestra el PUC dinámicamente, el controlador se ubica dinámicamente y determina el número de controladores en la red necesarios para soportar la carga. Los autores proponen un algoritmo de colocación del controlador, combinado un algoritmo de administración de flujo dinámico llamado LiDy, y los resultados muestran una menor latencia entre el controlador y los dispositivos de reenvío.

2.3.6. Métodos Heurísticos

Un algoritmo es un conjunto de instrucciones o pasos utilizados para realizar una tarea o resolver un problema [33]. Formalmente, un algoritmo es una secuencia finita de operaciones que se llevan a cabo sin ambigüedades, cuya acción ofrece una solución a un problema.

Los algoritmos heurísticos son aquellos que producen soluciones sin ninguna garantía de optimización y, a su vez, generalmente tienen un tiempo de ejecución mucho más corto [48].

Otra opción son las técnicas exactas, como LP (Programación lineal, que maximiza o minimiza una función lineal llamada función objetivo) [49], también la técnica para solucionar problemas denominada como PL (Programación lineal) donde algunas de las variables o todas están limitadas a ser enteros [50]. Los algoritmos exactos son muy lentos porque examinan todos los resultados posibles y esta acción requiere mucho procesamiento informático.

Dentro del grupo de algoritmos heurísticos, está el grupo de métodos metaheurísticos, que imitan fenómenos simples observados en la naturaleza y que parecen estar asociados con la inteligencia artificial [51]. Estos algoritmos intentan adaptar el comportamiento de diferentes especies a soluciones de problemas altamente complejos a través de la optimización. Los más representativos se enumeran a continuación:

- Algoritmos evolutivos (genéticos): basados en modelos biológicos que emulan el proceso natural de evolución.
- Algoritmos basados en el comportamiento de las comunidades de hormigas, abejas.
- Algoritmo de recocido simulado, que representa una forma efectiva y general de optimización. Es útil para encontrar óptimos globales en presencia de grandes cantidades de óptimos locales.

- Búsqueda heurística Tabú, que utiliza estructuras de memoria, cuando encuentra una solución potencial, se marca como "tabú" y el algoritmo no va a calcular de nuevo esta solución [52].
- Método aleatorio, que se emplea un grado de aleatoriedad en el algoritmo como parte de su lógica [53].
- Los sistemas multiagente, se conocen como un sistema computarizado que se compone de múltiples agentes inteligentes que interactúan. Los sistemas multiagentes pueden determinar problemas que son difíciles de resolver para un agente individual o un sistema monolítico [54].

En general, los métodos exactos son responsables de buscar y garantizar un óptimo local, mientras que los métodos metaheurísticos tienen mecanismos específicos para alcanzar un óptimo global, pero no garantizan ese alcance. Sin embargo, proporcionan una solución aceptable en un tiempo de cálculo razonable.

2.3.6.1 Algoritmo de enfriamiento simulado

Inventado en 1983, utiliza un enfoque similar a la escalada, pero a veces acepta soluciones que son peores que la actual. La probabilidad de tal aceptación disminuye con el tiempo.

El algoritmo de recocido simulado se basa en los principios de la termodinámica y el proceso de recocido del acero [34].

Considerando los problemas de optimización combinatoria como problemas que buscan un óptimo global, se pueden incluir procedimientos de búsqueda estocástica, como lo son alternativas heurísticas.

El nombre de "recocido simulado" se justifica por el temple o enfriamiento controlado con el que se producen sustancias específicas; Es el caso, por ejemplo, de la cristalización del vidrio o del recocido del acero [34].

Inicialmente, a temperaturas muy altas, se produce una amalgama líquida en la que las partículas se configuran aleatoriamente. El estado sólido se caracteriza por tener una configuración concreta de energía mínima (el mínimo global). Para lograr esta configuración, es necesario enfriar la amalgama lentamente, ya que

un enfriamiento repentino paraliza el proceso y alcanzaría una configuración diferente a la buscada (un mínimo local que no sea el mínimo global).

Las diferentes configuraciones que se pueden obtener se corresponden con las diferentes soluciones en el problema de optimización combinatoria, y el óptimo es el mínimo global. Puede ver el templado simulado como una variación de la simulación de Monte Carlo, cuyo estudio se basa en la simulación del comportamiento de una colección de átomos a una temperatura específica.

2.3.6.2 Tabú

La búsqueda de tabú amplía la idea de evitar el óptimo local mediante el uso de estructuras de memoria. La búsqueda tabú, a diferencia de las meta-heurísticas analizadas anteriormente, incorpora la memoria de las soluciones que han llevado a la última solución antes de decidir cuál es la siguiente. Se basa en los principios generales de la Inteligencia Artificial y fue desarrollado independientemente por Glover (1986) y Hansen (1986) [38].

2.3.6.3 Colonia de hormigas

Principalmente hay dos tipos de técnicas que hablan de sistemas comunes autoorganizados, la primera es la optimización de las Colonias de hormigas conocidas como CoH y el enfoque de optimización de enjambres de partículas (OEP), ambas introducidas en 1989 [55]. CoH mueve las hormigas (artificiales) y el cambio de posición para encontrar el mejor camino de solución. La OEP resuelve problemas dimensionales (óptimo local) cambiando la posición de la partícula esporádicamente. La ventaja fundamental del uso del proceso de inteligencia de enjambre es que tolera el problema local óptimo.

Los algoritmos CoH (Optimización de colonias de hormigas) son modelos inspirados en el comportamiento de colonias de hormigas reales. Los estudios realizados para explicar cómo los animales casi ciegos, como las hormigas, pueden seguir la ruta más corta en su camino de ida y vuelta entre la colonia y una fuente de suministro. Se debe

a que las hormigas pueden "transmitirse información" entre sí gracias al hecho de que cada una de ellas, al moverse, deja un rastro de una sustancia llamada feromona a lo largo del camino seguido.

Por lo tanto, una hormiga mueve el camino al azar, luego las otras hormigas siguen la trayectoria del camino a través de la feromona. Estos, a su vez, están dejando su feromona en el camino recorrido y por lo tanto lo hacen más atractivo, ya que ha reforzado el rastro de la feromona. Sin embargo, la feromona también se evapora con el tiempo, lo que hace que la trayectoria de la feromona sufra y se debilite. De esta manera, el crecimiento de las trazas es limitado, por lo que la solución adoptada podría corresponder a un óptimo local.

En consecuencia, hay una retroalimentación, donde el éxito está garantizado si el número de hormigas que viajan en una ruta anterior que ya viajó es más significativo.

El primer algoritmo de este tipo se denominó "Sistema de hormigas" para resolver el problema del viajero [56], obteniendo resultados no muy afortunados, pero que causaron la curiosidad de un gran número de investigadores que modificaban el algoritmo y aplicaban esta técnica a un conjunto de problemas.

Los algoritmos de optimización basados en colonias de hormigas se han aplicado fundamentalmente a problemas de optimización combinatoria. Dentro de este conjunto de problemas, se han utilizado en la resolución de problemas de optimización combinatoria NP-hard [57] en los que las técnicas clásicas ofrecen resultados no muy convincentes y problemas mínimos de ruta en los que la instancia del problema varía con el tiempo.

Como ya se mencionó, el primer problema en el que se aplicaron estos tipos de algoritmos fue el Problema del vendedor ambulante, una instancia de un problema NP-duro, que también incluye la resolución de una ruta mínima. Este problema se ha convertido en un estándar para probar en modelos post-iniciales propuestos por Bonabeau et al. (1999) [41].

Algoritmos de evolución

Los algoritmos evolutivos (EA) trabajan en la convergencia prematura y tienen en cuenta, al mismo tiempo, una serie de soluciones. Los EA se basan en la estructura evolutiva de la vida, teniendo en cuenta características como la selección, la reproducción y la mutación. Las soluciones se crean al azar o por selección, luego las soluciones se optimizan [58]. En la siguiente parte se dará una explicación del método genético.

2.3.7. Método Genético

Los algoritmos genéticos basados en los mecanismos de la genética y la selección natural. Como se verá más adelante, es la única metaheurística que funciona simultáneamente con dos conjuntos de soluciones factibles, que los considerará como individuos de una población que cruza, se reproduce y puede incluso mutar para sobrevivir. Fueron introducidos por Holanda (1975) en su obra Adaptación en sistemas naturales y artificiales [59]. El objetivo de su investigación era doble: por un lado, explicar con rigor los procesos de adaptación natural en los seres vivos; y por otro lado, diseñar programas informáticos basados en estos mecanismos naturales. La naturaleza nos enseña que los mecanismos de adaptación y supervivencia funcionan bien en los seres vivos.

Descripción del algoritmo

Operadores Básicos

Con algoritmos genéticos, estas soluciones se consideran como individuos de una especie que evolucionan para adaptarse y mejorar esa especie. La calidad de los individuos y, en última instancia, de la especie, medida por la función de aptitud [60]. En este trabajo se consideraron operadores de selección y mutación.

Operador de selección

No todos los individuos sobreviven por un tiempo específico, por ejemplo, hasta la madurez; Sólo los más aptos lo harán, es decir, aquellos que tengan una mejor

función de aptitud. Una posibilidad para hacer esto sería considerar que solo la mitad de los individuos sobrevive, con la probabilidad de supervivencia de cada uno proporcional a su función física.

Operador de Mutación

Tras el cruce, se produce la mutación. Respecto a la evolución, la mutación se manifiesta extraordinariamente, nada estándar. Las mutaciones tienden a ser beneficiosas en promedio porque contribuyen a la diversidad genética de la especie. También evitan que las soluciones de la población se vean limitadas por un óptimo local. Por lo tanto, la mutación consiste en modificar aleatoriamente genes específicos, teniendo en cuenta la probabilidad de mutación establecida previamente. La mutación depende de la codificación y la reproducción. Si se abusa de la mutación, puede caer en el uso del algoritmo genético como una simple búsqueda aleatoria. Por lo tanto, antes de aumentar las mutaciones, es recomendable estudiar otras soluciones que proporcionen diversidad a la población, como aumentar el tamaño de la población o garantizar la aleatoriedad de la población inicial.

Cuando hay genes de un cromosoma cambiado es el resultado de la mutación. Si la descendencia es idéntica a la reproducción posterior, significa que no hay mutación. En caso de alteración, se ajusta un segmento del progenitor. Cuando hay un 100% de posibilidades de mutación es porque el cromosoma ha cambiado completamente.

Para finalizar este capítulo se muestra un resumen de 65 publicaciones que hablan sobre el problema de ubicación del controlador. Como se observa en la tabla 2.1 la propuesta para abordar este problema muestra la cantidad de publicaciones que utilizaron los diferentes autores para abordar el problema de ubicación del controlador.

Número de publicaciones	Cómo abordaron el PUC
2	Número de controladores
3	Costo
4	Capacidad del controlador
5	Balanceo de carga
16	Resiliencia/confiabilidad
17	Latencia
18	Multiobjetivo

Tabla 2.1. Número de publicaciones realizadas respecto a cómo se abordó el PUC.

Las redes definidas por software transforman el concepto tradicional de cómo implementar, gestionar y/o modificar una red, ya que dividen el plano de control y datos; Donde el plano de control decide cómo se distribuye el tráfico mientras que el plano de datos reenvía los paquetes de acuerdo con la decisión tomada por el plano de control. Un elemento fundamental en una RDS es el controlador ya que gestiona el control de flujo y permite crear las políticas de cómo será administrada la red. Las investigaciones realizadas para el problema de ubicación del controlador muestran que se han implementado modelos exactos y heurísticos evidenciando que el problema se abordó en su mayoría de veces considerando métricas de latencia y multiobjetivo.

3. Algoritmo genético

Las redes definidas por software representan un nuevo paradigma para las redes de comunicaciones, ya que prometen una mayor eficiencia en el uso de las mismas redes a un costo económico menor, a raíz de este nuevo paradigma se desarrollan diferentes técnicas para que las RDS sean más eficientes, este capítulo expone un algoritmo genético aplicado a una RDS considerando el modelo matemático tenido en cuenta para el modelamiento del problema, los nodos, enlaces, controladores y la función a optimizar que para este caso es el costo económico. También muestra el operador de mutación y selección necesarios para que se cumpla la condición de un algoritmo genético.

Se escogió un algoritmo genético porque su tiempo de procesamiento es mucho menor respecto a las técnicas exactas las cuales tienen mayor duración en adquirir un resultado.

También opera de manera simultánea con varias soluciones y no de manera secuencial (búsqueda exhaustiva) o de memoria programada (Tabú). Se diferencia de otros algoritmos heurísticos en el sentido de que la solución no se queda atrapada en mínimos locales tal como lo hace el algoritmo de enfriamiento simulado.

3.1. Modelo de Redes Definidas por Software

En este trabajo se considera una RDS como una red WAN de datos compuesta por switches, controladores y enlaces. Como se evidencia en el anterior capítulo de este trabajo, existen dos planos, el plano de datos compuesto por switches y el plano de control compuesto por controladores. Se utiliza control en banda en el cual la red de control se superpone con la red de datos, en otras palabras, el tráfico de control puede compartir la misma red con el tráfico de datos, como se observa en la figura 3.1.

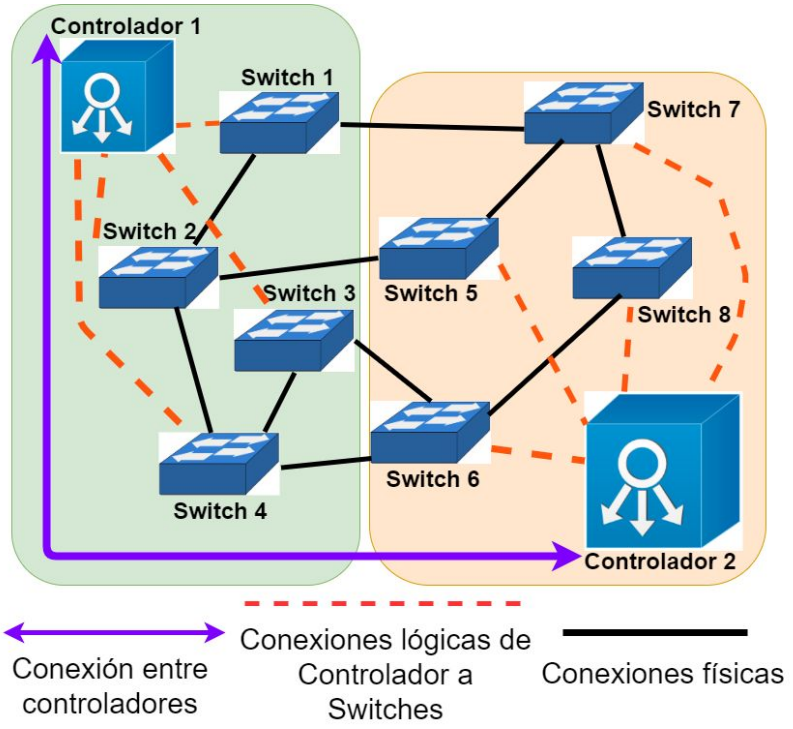


Figura 3.1. Red Definida por Software. [Gráfica de elaboración propia]

Para un switch del plano de reenvío de paquetes reciben un paquete, si el destino del paquete no se encuentra en la tabla de enrutamiento, entonces el switch envía una solicitud de establecimiento de flujo (SEF) al controlador que los gobierna.

La RDS se modeló como un grafo representado por:

$$S = (N , J)$$

Ecuación 1. RDS representada como un grafo.

Donde N representa los switches o nodos y J los enlaces. Cada switch en el grafo puede llegar a ser un controlador el cual se denota de la forma ($C \subseteq N$)

Un switch puede convertirse en un controlador activando una licencia de software. Esto sucede cuando los controladores están sobrecargados, lo cual quiere decir que en un escenario en el cual exista un gran número de SEF el rendimiento del controlador se degrada. Este hecho afecta el tiempo de procesamiento medio para cada solicitud, si el número de SEF se duplica, el tiempo de procesamiento del controlador también se duplica.

Para formular el modelo matemático, se asume lo siguiente:

- La latencia máxima permitido en la red para las comunicación entre switch-controlador (latencia de configuración de flujo máximo) y la latencia máxima permitida entre controladores.
- El costo de asignar una licencia para ejecutar un controlador o switch.
- Las capacidades de hardware de los switches para ejecutar las licencias de los controladores y manejar la configuración del flujo.
- La latencia de reenvío y el costo de usar un enlace que pertenece a un proveedor de servicios entre cada par de nodos.

Teniendo en cuenta la información anterior, se formula el problema de la siguiente manera:

Se tiene un conjunto de nodos (N) que en conjunto conforman enlaces (J). El grafo o red se conforma por enlaces (J) y nodos (N). En una red SDN se necesitan las posibles ubicaciones del controlador (C), donde (C) hace parte del conjunto de nodos (N). Se considera el costo de tener una licencia en un controlador (Z) y el costo de un camino del proveedor de servicio desde un nodo (N) a otro nodo (N) dado por $\lambda^{p}_{n,n'}$. También se considera el número de requerimientos presentes en un controlador βc y el tiempo promedio en el cual se maneja un número de requerimientos de instalación de flujo en un controlador ωc. La latencia de reenvío dada en milisegundos entre nodos y entre controladores para un proveedor de servicio se da por *p*.

De acuerdo con la teoría de colas basada estudios matemáticos dentro de un sistema, existen diferentes disciplinas de de cola, la cual indica el orden de selección de miembros para atender el servicio. En este trabajo maneja la disciplina de cola "FIFO" la cual viene de las siglas en inglés "First In, First Out", indica que el primero en entrar es el primero en salir, es decir se respeta el orden de llegada para atender la solicitud, para este caso será (βc). Es decir no existirá ningún tipo de prioridad para ninguna SEF.

El tiempo promedio en el cual el controlador procesa la SEF (ρ_c) está determinado primero por el orden de llegada de la SEF a la cola del controlador (βc) y segundo por el retardo para el procesamiento de la SEF (ωc).

El elemento de red "Switch" puede tener también el rol de "controlador" es decir, sobre el mismo hardware se puede tener un switch o controlador. Dependiendo de las características del hardware, se determina el retardo para el procesamiento de la SEF (σ_c) . Entonces el tiempo promedio (ρ_c) está dado por $\beta c\, \omega_c$.

El total de SEF que recibe el controlador corresponde a la suma de las SEF enviadas por los switches que son gobernados por un mismo controlador como se observa en la figura 3.2.

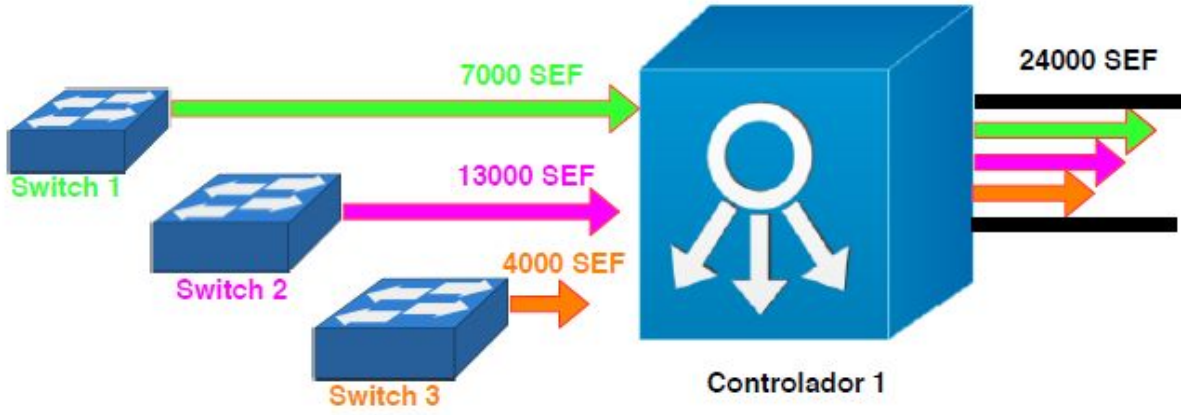


Figura 3.2. Solicitud de establecimiento de flujo (SEF) en un controlador.
[Gráfica de elaboración propia]

Los Switches envían sus paquetes SEF a través de los enlaces. Los paquetes son transportados por un proveedor de servicio ρ, donde el precio se representa como $\lambda^{p}_{n,n'}$ y la latencia como $l^{p}_{n,n'}$ donde n es el switch de origen y n' es el switch de destino al Switch. En este caso se tendrán *x* proveedores de servicio los cuales se representan en la figura 3 mediante líneas de colores.

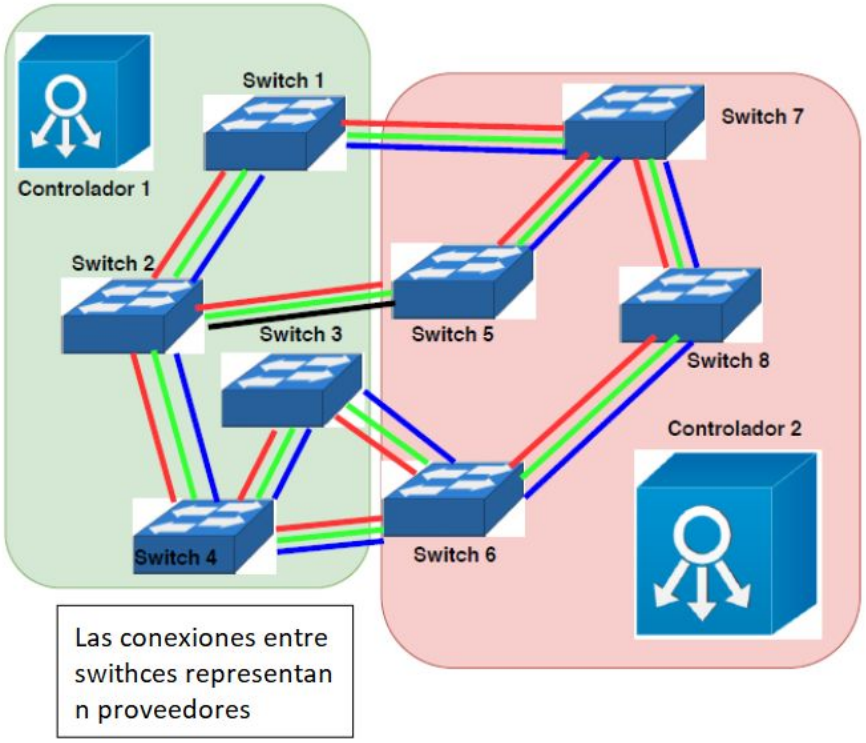


Figura 3.3. RDS con diversidad de proveedores.[Gráfica de elaboración propia]

El contexto de este trabajo se estructura en el escenario de cloud computing, lo cual facilita que los recursos de IT sean consumidos como servicio [61]. En este trabajo, los controladores son un software instalado sobre un Switch que se activan bajo una licencia en un tiempo determinado, a un precio establecido.

A continuación se presentan las restricciones y variables:

2) Restricciones:

- ϕ, la máxima latencia de configuración de flujo máximo (en milisegundos).
- η, la máxima latencia entre controladores (en milisegundos).

3) Variables

- x_c, es una variable binaria tal que $x_c = 1$ sí y solo sí un controlador se despliega en una posible ubicación de controlador con $c \in C$.
- $k_{u,c}$, es una variable binaria tal que $k_{u,c} = 1$ sí y solo sí un nodo $u \in N$ es asignado al controlador $c \in C$.

- $z_{n,n'}$, es una variable binaria tal que $z_{n,n'}$=1 sí y solo sí un nodo $u \in C$ y un nodo $v \in C$.
- ρ_c, el retardo de procesamiento (en milisegundos) para un flujo de configuración con $\rho_c = \beta_c \omega_c$, $\forall c \in C$.
- δ_u, representa la latencia entre un nodo $u \in N$ *y su controlador* $c \in C$.

A) Función de costo

El objetivo es minimizar los costos asociados al control de la red, los costos están dados por:

- El costo de correr licencias en los controladores:

$$Cc(x) = \sum_{c \in C} Bx_c$$

Ecuación 2. Costo de las licencias en los controladores.

- El costo de enviar información de control entre un switch y su controlador asignado.

$$Cs(k) = \sum_{u \in N} \sum_{c \in C} \sum_{p \in P} \lambda^{p}_{n,n'} k_{u,c}$$

Ecuación 3. Costo de envío de información de un SW a un Controlador

- El costo de enviar información de control entre los controladores

$$Cw(z) = \sum_{n \in N} \sum_{n' \in C} \sum_{p \in P} \lambda^{p}_{n,n'} k_{u,c}$$

Ecuación 4. Costo de envío de información entre controladores.

B) Modelo

El problema de ubicación del controlador se modela de la siguiente manera:

$$\textit{Función objetivo a minimizar} = (C_c(x) + C_s(k) + C_w(z))$$

Ecuación 5. Función objetivo a minimizar.

3.2. Algoritmo Genético para la solución del PUC

El algoritmo genético es una forma de tratar el PUC . La RDS que se modeló anteriormente fue la usada por el algoritmo genético. El costo de la solución representa la calidad de la misma, puesto que entre menor el costo, la solución será mejor.

Se consideraron los siguientes parámetros para desarrollar el algoritmo genético:

- El algoritmo genético parte de un conjunto de soluciones iniciales, con un tamaño de población limitada. En este trabajo, se tomó una población de 10 soluciones.
- La soluciones iniciales conocidas como primer población o primer generación generan una mutación aleatoria del 10% de la población total, incrementando el tamaño de la población inicial.
- La población de tamaño original debe mantenerse, por tal razón se elimina el 10% de las peores soluciones o elimina el 10% de la población de manera aleatoria.
- Se repite el proceso de mutar soluciones por cada generación hasta llegar hasta la generación 200 con el fin de determinar qué tan óptima es la solución.

Genes

Los genes son un conjunto de parámetros que agrupados forman un conjunto de soluciones.

Los atributos son las características de los elementos que conforman la red, en este trabajo los elementos que contienen atributos son el controlador, el switch, la red y los enlaces. Estos atributos pueden cambiar. Para los que se muestran como “SI” en las tablas que se exponen a continuación, los atributos tendrán sustituciones, por ejemplo en el atributo “Switches que controla” el controlador gobernará 5 switches, 10 switches ó n switches, si el atributo no cambia, quiere decir que mantendrá siempre mantendrá su característica original, por ejemplo el atributo “Id del controlador” nunca cambiará. En las siguientes tablas se muestran los atributos que pueden cambiar y no pueden cambiar.

Controlador

Atributo	Cambia
Id del Controlador	NO
Desplegado en	SI
SW Controlado	SI
Estado	SI
Recibir Solicitud	SI
Retraso de procesamiento	SI

Tabla 3.1. Atributos del controlador y cambios.

Switch

Atributo	Cambio
Id del Switch	NO
Configuración de flujo	SI
Controlado por	SI
Retardo de procesamiento	SI

Tabla 3.2. Atributos del switch y cambios.

Red

Atributo	Cambio
Costo del controlador	NO
Costo del enlace entre Controlador y Switch	SI
Costo del enlace entre controladores	SI
Costo total	SI
Lista de Switches	NO
Controladores	SI
Lista de enlaces	NO

Tabla 3.3. Atributos de red y cambios.

Enlace

Atributo	Cambio
Id del enlace	NO
Retardo	SI
Costo($)	SI
Id del proveedor	NO
Id origen	NO
Id destino	NO
Estado	NO

Tabla 3.4. Atributos de Enlace y cambios.

Cromosoma

Es una solución propuesta al problema que el algoritmo genético está tratando de resolver.

En este caso, el cromosoma es la red que se compone de controladores, switches y enlaces. Las diferentes ubicaciones del controlador o los controladores que conforman un conjunto de soluciones.

Solución

Una solución es una RDS conformada por uno o más controladores. Está red mostrará tres costos, el costo por cada licencia de controlador activada, el costo de los enlaces activos entre los switches y controladores y el costo entre los enlaces que comunican los controladores con licencia activa en la red.

Población:

La población es un subconjunto de soluciones en la generación actual. También se puede definir como un conjunto de cromosomas.

Primera generación

Es el primer conjunto soluciones aleatorias generadas por primera vez.

Proceso del Algoritmo genético

Como se observa en la figura 3.4 se observa el algoritmo genético en un alto nivel incluyendo los operadores de mutación y operador de selección, los cuales se explican a continuación.

A continuación se resume el proceso del algoritmo genético:

1. Se crea la primera generación (población original) compuesta por un número limitado de soluciones o cromosomas generados.
2. Se mutan las soluciones, creando una cantidad finita de soluciones nuevas, con una cantidad superior de soluciones a la población anterior.

3. La superpoblación se controla con el operador de selección, las soluciones más débiles(las que tienen mayor costo) se eliminan. Al final de este proceso, el tamaño de la población es el mismo que el de la población original.
4. Se genera la segunda generación la cual contiene el mismo número de soluciones o cromosomas visto en 1.
5. Se vuelve a 2 para realizar el mismo proceso visto anteriormente. Se continua iterando y así se encontrarán las mejores soluciones. El proceso termina hasta finalizar con el total de iteraciones.

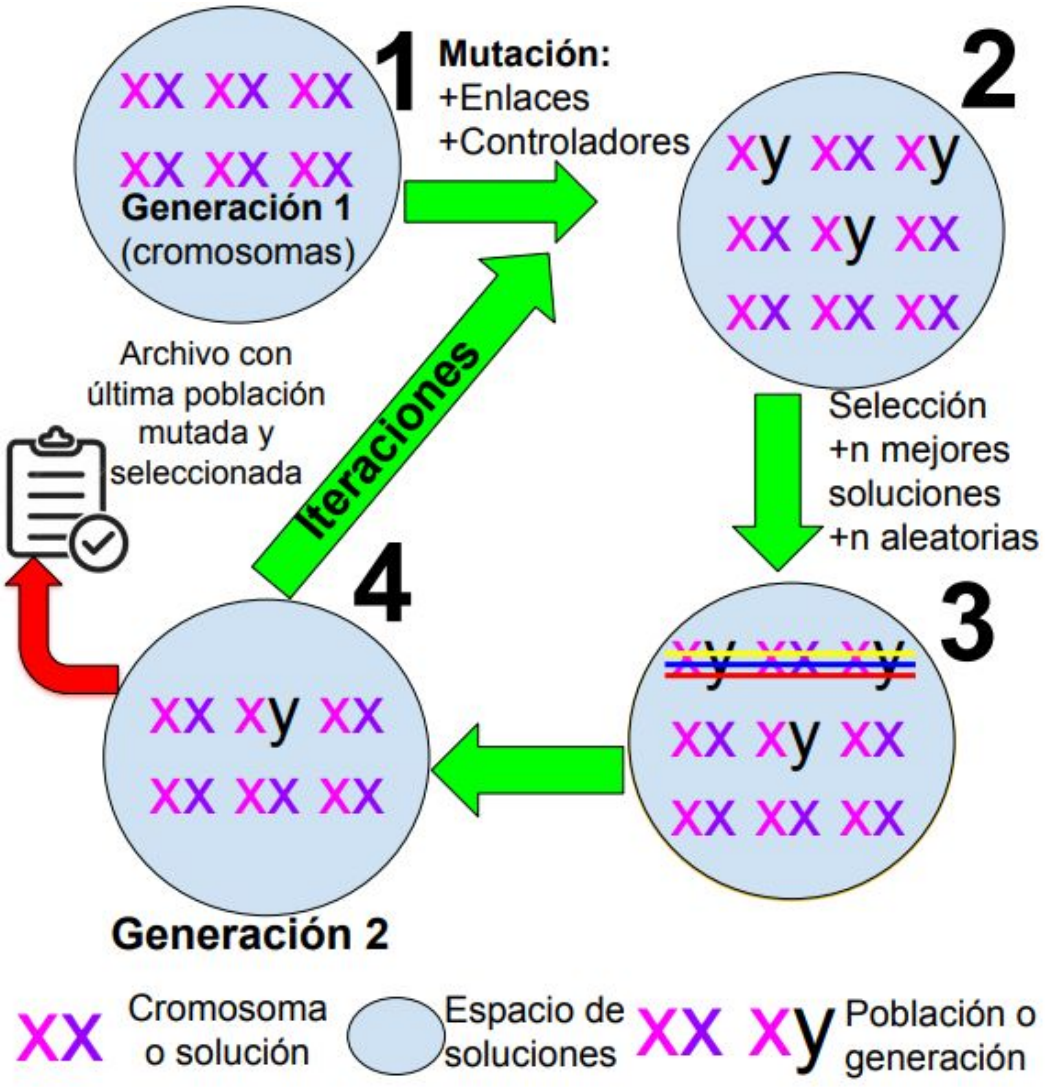


Figura 3.4. Funcionamiento del algoritmo genético en un alto nivel.[Gráfica de elaboración propia]

Se incluyen los operadores de mutación y selección.

Operador de Mutación y operador de selección

Un operador genético guía al algoritmo hacia una solución para un problema dado. En este trabajo se considera el operador de mutación y el operador de selección, los cuales se muestran a continuación.

Operador de Mutación

La mutación se define como un ajuste aleatorio en el cromosoma para obtener una nueva solución.

Para este trabajo, la mutación se encuentra en dos órdenes:

1. Mutar enlaces
2. Mutar controladores

Operador de Selección

Este operador mantiene el tamaño de la población original [62], por lo tanto el operador elimina los excesos de población generado por las mutaciones, se usa un operador de selección elitista el cual elimina las soluciones más débiles generación tras generación. De esta forma se garantiza que después de cada mutación, el tamaño de la población se conserve respecto al original.

3.3. Restricciones consideradas para generar soluciones

El modelo considera dos restricciones para que se generen las respectivas soluciones, las restricciones son divididas en restricciones del problema y restricciones de calidad.

A) Las restricciones son límites que se imponen en este trabajo para que se cumplan las condiciones de una RDS y se generen las soluciones, estas restricciones se mencionan a continuación:
    1. Un switch debe ser controlado solamente por un controlador.
    2. En un switch solo puede desplegarse un controlador.

B) Con el fin de asegurar la calidad en cada solución, a continuación se muestran las restricciones correspondientes a la latencia:

1. La máxima latencia para establecer un flujo a través de un SEF (Φ) se ve en la siguiente ecuación:

$$2\lambda^{p}_{n,n'} + \rho_c \leq \Phi$$

Ecuación 6. Máxima latencia para establecer un flujo a través de una SEF.

lo cual quiere decir que la latencia de ida y regreso más el tiempo promedio de procesamiento de la SEF debe ser inferior al valor máximo para establecer el SEF.

2. La máxima latencia entre controladores (η) se da por la siguiente ecuación:

$$2 \sum_{\mu \varepsilon C} L^{p}_{n,n' z_{n,n'}} \leq \eta,$$

Ecuación 7. Máxima latencia entre controladores.

donde, C representa el conjunto de controladores, si el enlace conecta dos controladores. La restricción asegura que la latencia de ida y vuelta entre controladores que están unidos por un enlace será igual o menor al retardo máximo establecido.

3. Se garantiza que el retardo entre sw y controlador será 0 siempre y cuando el controlador se instale sobre el mismo switch.

3.4. Soluciones generadas por mutación

Las mutaciones se denominan como “movimiento” y serán las encargadas de generar nuevas soluciones.

Se llevaron a cabo dos tipos de movimientos para generar nuevas soluciones aleatorias, los cuales se muestran a continuación.

3.4.1. Soluciones generadas por movimiento de controladores

La población aleatoria que se genera inicialmente debe cumplir con las restricciones vistas previamente. El movimiento de controladores identifica cuales controladores están activos y cuales inactivos en una o varias soluciones de la población [63], un controlador gobierna uno o más switches los cuales son almacenados en una lista.

Un controlador desplegado en un switch cambia de estado activo a inactivo e inversamente un switch que tiene su controlador desactivado pasa a activarse, y heredará la lista de switches gobernados por el switch que se apagó anteriormente, este proceso de cambio de controlador se realiza respetando las restricciones mencionadas anteriormente

En el siguiente diagrama de flujo de la figura 3.5 se muestra muestra el movimiento de controladores:

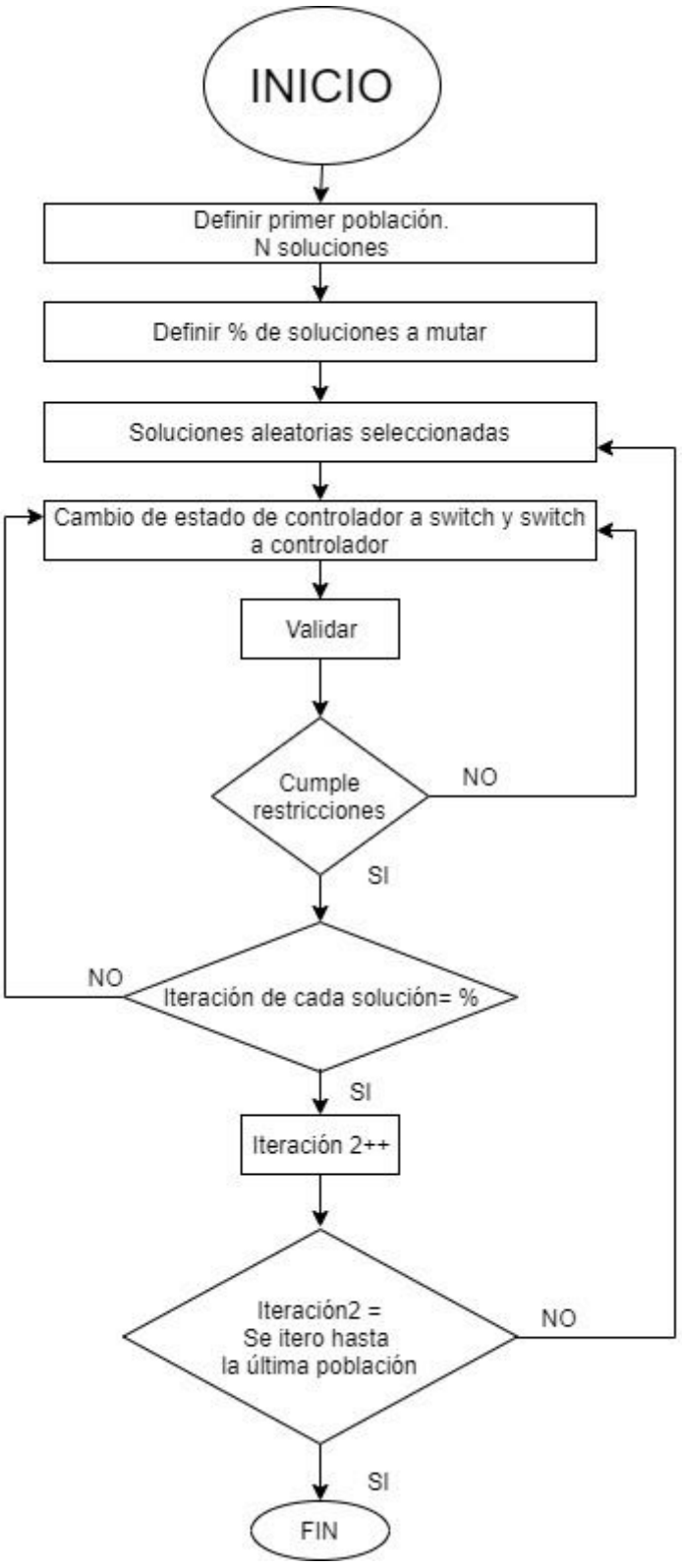


Figura 3.5. Diagrama de flujo de mutación por movimiento de controladores.

3.4.2. Soluciones generadas por movimiento de enlaces

El movimiento de enlaces se fundamenta en encender y apagar enlaces entre dos nodos directamente conectados. Un nodo siempre se encontrará conectado directamente a otro nodo a través de n enlaces, cada enlace representa un proveedor de servicios, los proveedores de servicios manejan costos diferentes y entre dos nodos que se encuentran directamente conectados siempre existirá un enlace activo. Este movimiento se caracteriza apagar un enlace activo y encender un enlace inactivo al azar [63]. Solo un enlace estará activo cuando exista una conexión directa entre dos nodos.

En el siguiente diagrama de flujo de la figura 3.6 se muestra el movimiento de enlaces:

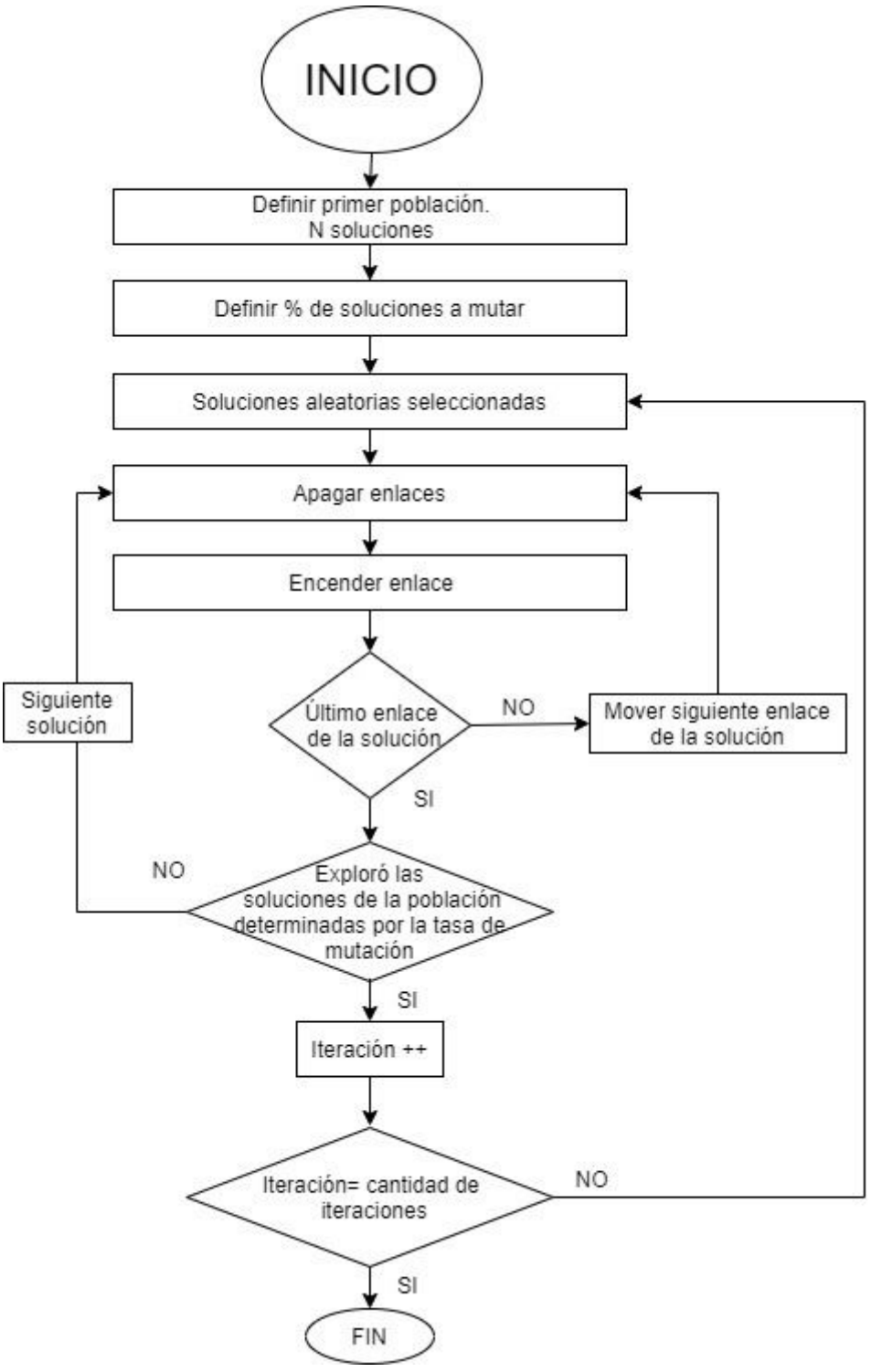


Figura 3.6. Diagrama de flujo de mutación por movimiento de enlaces.

3.4.3. Operador genético de selección

Una vez realizada la mutación de controladores o la mutación de enlaces, se genera una nueva población que contiene soluciones adicionales a la población original, generando una sobrepoblación, estos excesos de población se controlan mediante el operador genético de selección eliminando las peores soluciones, solo las soluciones fuertes con menor costo económico sobreviven, al final de este operador de selección, se contará con el tamaño original de la población inicial.

A continuación se muestra el diagrama de flujo de la figura 3.7 se explica el operador genético de selección:

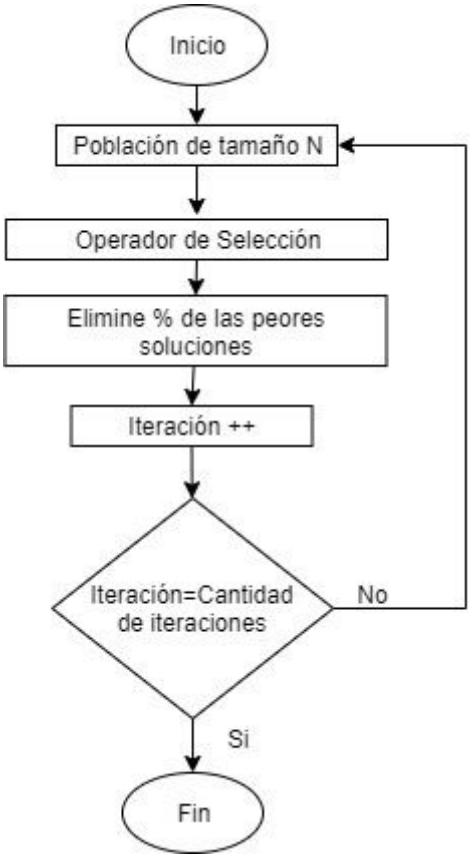


Figure 3.7. Proceso genético de selección.

En este capítulo el modelo genético fue elaborado para una RDS considerando enlaces, controladores y nodos a utilizar con el fin de encontrar la función objetivo que

se conforma por la sumatoria del costo de activar una licencia en el/los controlador(es), el costo de envío de información de un SW a un controlador, y el costo de de envío de información entre controladores. El algoritmo genético utiliza los operadores de selección y mutación para encontrar las mejores soluciones.

4. ANÁLISIS Y RESULTADOS

Esta sección explica cómo fue implementado el algoritmo genético del capítulo 3. También se "presenta" el desempeño de dicho algoritmo. El algoritmo fue evaluado con algunas topologías de la base de datos del "*Topology Zoo*" que fueron adaptadas al modelo RDS explicado en la sección 3.1.

4.1. Implementación del algoritmo genético

4.1.1. Insumos del algoritmo genético

A continuación se mostrarán los diferentes insumos requeridos para el desarrollo del algoritmo genético.

1. **Carga de topologías:**

La información suministrada por la base de datos del *Topology Zoo* [64] muestra la relación de los nodos como grafos representados en archivos de formato GML. Los nodos se encuentran unidos por aristas.

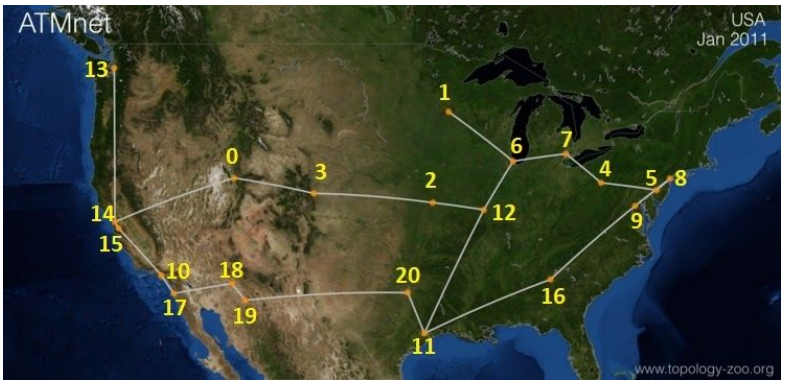


Figura 4.1. Topología ATMnet. [Gráfica tomada del Topology Zoo y modificada personalmente]

Para iniciar, se toman los nodos que se encuentran directamente conectados. Cada nodo cuenta con una ubicación expresado en términos de latitud y longitud, estas coordenadas brindadas por cada nodo se aplican a la ecuación de Haversine [66]. Con la ecuación de Haversine se calcula la distancia entre dos puntos basándose en

coordenadas angulares de latitud para norte o sur y longitud para este u oeste, los datos de las coordenadas son brindados por el archivo GraphML del *topology Zoo.*

$$d = 6371 * Acos(Cos(90 - La_1) * Cos(90 - Lo_1) + Sin(90 - La_1) * Sin(90 - Lo_1) * Cos(La_2 - Lo_2)$$

Ecuación 8. Ecuación de Haversine para el cálculo de distancia entre dos puntos considerando latitud y longitud.

El medio de transmisión que se tuvo en cuenta para este trabajo es la fibra óptica y se considera la velocidad de la luz en la fibra óptica la cual se expresa en la siguiente ecuación:

$$v = 2 * 10^8 \frac{m}{s}$$

Ecuación 9. Velocidad de propagación de la luz en una fibra óptica.

Una vez se tiene la distancia y velocidad se realiza el cálculo de la latencia, expresado en la siguiente ecuación:

$$L = \frac{d}{v}$$

Ecuación 10. Cálculo de la distancia entre dos nodos directamente conectados.

El resultado obtenido se guarda en un archivo de texto plano (plain text), el cual contiene la información de los dos nodos directamente conectados y la latencia existente entre ellos:

Nodo X	Nodo Y	Distancia entre nodos
0	3	2,984315855
0	14	4,755967249
1	6	2,856848158
.	.	4,45902434
.	.	1,936203746
.	.	2,06967193
17	18	2,402552282
18	19	0,8674410517
19	20	6,623102988

Tabla 4.1. Latencia generada entre nodos. Basado en la figura 4.1

2. Cálculo de las latencias entre nodos que no están directamente conectados

Una vez se tiene el archivo de texto plano con la latencia entre los nodos directamente conectados, se toma esta información y se crea un diccionario de datos que es una colección de valores de datos el cual admite almacenar valores enteros de cadena o listas. Con esta información se determina el camino más corto entre dos nodos con un algoritmo de caminos mínimos o mejor conocido como Algoritmo de Dijkstra [34], el resultado de este proceso es la ruta de nodos con la menor latencia.

3. Creación de las topologías para la RDS

Cuando se generan las latencias de cada pareja de nodos, son creados los detalles de la RDS en un archivo de texto plano. Esta configuración se muestra a continuación:

```
LC0_0,"LC0_0",0,0,"Claro","S0","S0"
LM0_0,"LM0_0",0.0,0,"Movistar","S0","S0"
LT0_0,"LT0_0",0.0,0,"Tigo","S0","S0"
LC0_1,"LC0_1",0.2326,100,"Claro","S0","S1"
LM0_1,"LM0_1",0.25586000000000003,75,"Movistar","S0","S1"
LT0_1,"LT0_1",0.27912,60,"Tigo","S0","S1"
```

Figura 4.2 Formación de una RDS.

Extraída de un archivo de un texto plano muestra cómo se conforma la RDS.

Acorde con la figura 4.2 se evidencia ”Id Link” el cual representa la conexión entre dos nodos directamente conectados, se observa que existen n criterios de selección o proveedores. Una vez se tiene la topología de la RDS, se toma el nodo inicial o nodo 0 y se empieza a calcular la menor latencia respecto a los demás nodos de la red gracias al algoritmo de Dijkstra, en este punto se añaden las características que hacen parte de los switches y controladores. Como se vio en el capítulo anterior un switch puede llegar a transformarse en controlador, por dicha razón sin excepción alguna, los controladores estarán desactivados.

4.1.2 Topologías simuladas con el algoritmo genético

El algoritmo fue simulado en un computador Intel® Core™ i7-8750H Processor 2.2 GHz con 32 Gb de ram. Las topologías simuladas fueron las siguientes:

Topología	Número de nodos
ATMnet	21
Packetexchange	21
YorkDataServices	23
IntegraTelecom	27
Darkstrand	28
Digex	31
Bics	33
NetworkUsa	35
PalmettoNet	45
Bellcanada	48

Tabla 4.2. Topologías simuladas

4.2. Algoritmo genético

Para la simulación del algoritmo genético se consideraron los siguientes parámetros:

Latencia de configuración de flujo máximo (LCFM): 50 ms, 75 mS, 100 ms ó 200 ms, 225 ms, 250 ms ó 275 ms, 300 ms, 325 ms. Esto depende de la topología que se trabaje.

El LCFM, es un parámetro que indica el tiempo máximo para establecer un flujo desde un origen un destino entre dos nodos que no sean controladores. Si no se cumple con esta condición, el algoritmo descarta esta solución y pasa a la siguiente iteración. Este valor está limitado por el tamaño de la topología, para las topologías ATMNet, YorkDataServices, Digex, Bics, CRL, NetworkUsa y PalmettoNet se tuvo en cuenta un valor de 50ms, 75ms y 100ms que se representa en milisegundos. Para la topología

Packetexchange se consideraron los valores de 275ms y 300ms. Para la topología de Darkstrand se tomaron los valores de 200ms, 225ms y 250ms y finalmente para Bellcanada se tomó 200ms, 225ms y 250ms.

Máxima latencia entre controladores (MLEC): 50 ms, 75 ms, 100 ms ó 200 ms, 225 ms, 250 ms ó 275 ms, 300 ms, 325 ms. Esto depende de la topología que se trabaje.
El MLEC, es una restricción que limita la calidad de las soluciones, indica el tiempo máximo en el cual dos controladores tardaran en comunicarse, si este tiempo es superior, no cumplirá la restricción del algoritmo y entonces se pasará a la siguiente iteración. Este valor está limitado por el tamaño de la topología, para las topologías ATMNet, YorkDataServices, Digex, Bics, CRL, NetworkUsa y PalmettoNet se tuvo en cuenta un valor de 50, 75 y 100 que se representa en milisegundos. Para la topología Packetexchange se consideraron los valores de 275 y 300. Para la topología de Darkstrand se tomaron los valores de 200ms, 225ms y 250ms y finalmente para Bellcanada se tomó 200ms, 225ms y 250ms.

PN: 1, 5, 10, 15
El porcentaje de nodo, expresa la cantidad de controladores que se tendrán en la topología, el porcentaje de nodo en este algoritmo es de 1, 5, 10 ó 15 y se aplica cada generación que contiene 10 soluciones, entonces por ejemplo si el PN es de 15 quiere decir que 1 ó 2 soluciones de la generación tendrán cambios.

TM: 0.5, 1
La tasa de mutación, indica la cantidad de mutaciones que se tendrán en la iteración, para este trabajo se maneja al 0.5 y 1. Por ejemplo si se tienen 10 soluciones en una generación y se realiza la mutación al 1, se tendrá una nueva población de 20 soluciones, de las cuales el algoritmo descarta las 10 soluciones más costosas.

Dependiendo de la combinación de los parámetros aplicados de LCFM, MLEC, PN y TM se obtendrán resultados, no todas las combinaciones posibles arrojaron soluciones ya que no cumplieron con las restricciones.

4.2.1. Resultados

Se establecieron 5 clases de resultados, los cuales se describen a continuación. En el anexo A se muestran los resultados para el total de topologías.

4.2.2. Costo Vs Iteración

Como se observa en la figura 4.3 se evidencia el costo máximo, costo promedio y costo mínimo cada vez que se encontraba una solución, se observa el decrecimiento de los costos a lo largo de cada iteración, lo cual significa optimización de la variable costo. Esta gráfica fue tomada de la topología "ATMnet", se consideraron los valores de LCFM:100, MLEC:75, PN:15, y TM:1. Esta gráfica se realiza para cada topología mencionada en este trabajo.

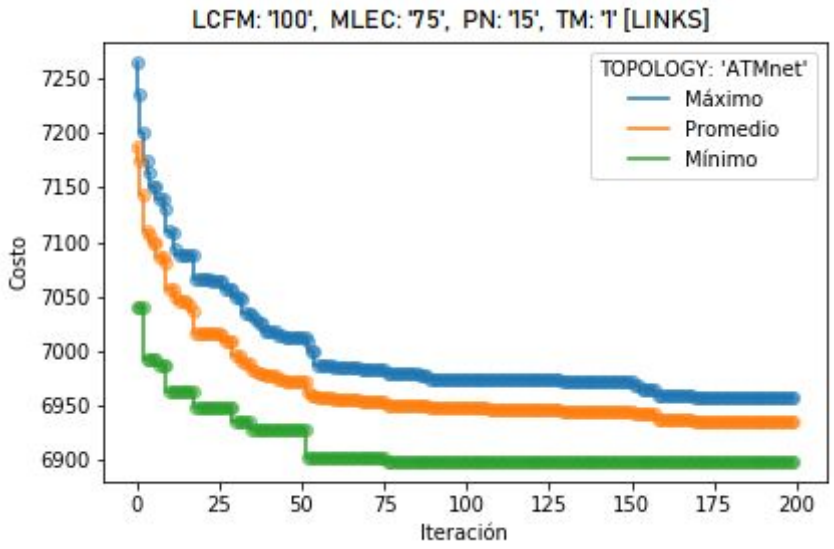


Figura 4.3. Costo de la topología ATMnet a lo largo de cada iteración.[Resultados al simular en python]

4.2.3. Topología inicial, topología optimizada vs Costo

Como se observa en la figura 4.4 el costo de la solución inicial que se genera aleatoriamente en cada topología, la leyenda que tiene el símbolo asterisco, muestra la solución óptima luego de la última iteración del algoritmo genético, se observa como los valores de “MIN, MEAN y MAX” disminuyen en las topologías que contienen el asterisco respecto a su muestra original. A mayor cantidad de nodos el costo final de

nodos es mayor a una topología que cuenta con menor cantidad de nodos. También se observa que no siempre se van a ver en la gráfica el total de las topologías estudiadas en este trabajo debido a que no cumplen con las restricciones mínimas. El costo mínimo sin optimizar no es muy diferente del costo mínimo optimizado, mientras que en el costo promedio y el costo máximo se ve una reducción del 5% al 8% del costo. Como se observa en la figura 4.4 se consideraron los parámetros de LCFM:100, MLEC:100, PN:1, TM:0.5

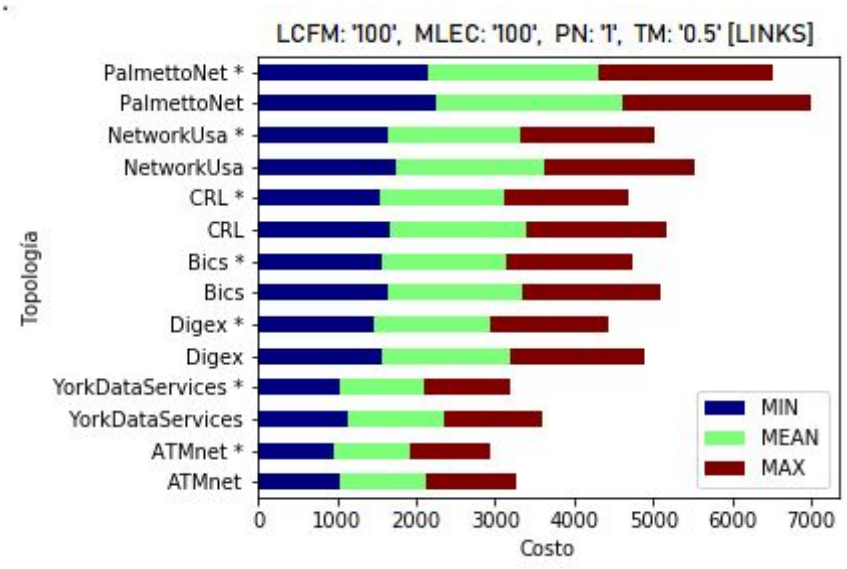


Figura 4.4. Costos de cada topología. [Resultados al simular en python]
El costo se evidencia antes y después de la optimización luego de 200 iteraciones.

4.2.4. Latencia de configuración de flujo máximo de cada topología Vs Costo

Como se observa en la figura 4.5 se evidencia la variación del costo total de cada topología de acuerdo con la “Latencia de configuración de flujo máximo”. Los espacios que se encuentran en color cercano al blanco indican que el costo es 0, lo cual quiere decir que en ese caso no existe solución de acuerdo con las restricciones mínimas. También se puede observar que no importa la cantidad de nodos de cada topología para determinar el costo de la solución, como se demuestra en la topología de "33 Bics" con "Latencia de configuración de flujo máximo de 50" en la cual el costo es 10.000 mientras que la topología "45 Palmetto Net" tiene una solución con costo cercano a 8000.

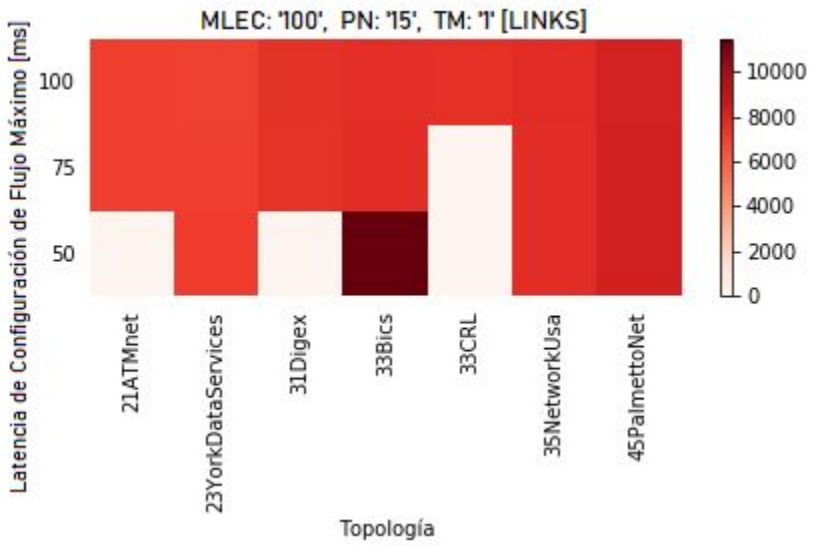


Figura 4.5. Costo de cada topología vs el LCFM.[Resultados al simular en python]
Para el LCFM se tomaron los valores de (50,75,100), con los parámetros de MLEC:100, PN:15, TM:1

4.2.5. Mutación de cada topología vs Costo

Como se observa en la figura 4.6 se evidencia la variación del costo total de cada topología de acuerdo con el “tasa de mutación”. Cuando el color es muy claro, el costo es 0, lo cual indica que esta topología no cumple con los parámetros mínimos para generar una solución, como se evidencia para la topología de “33 Bics” con una “tasa de mutación” del 50%”. Si cumple con las restricciones mínimas a pesar de que la “tasa de mutación” varía, el costo se mantiene. No todas las topologías que son parte de este trabajo serán vistas en la misma gráfica debido a que no cumplen con las restricciones mínimas.

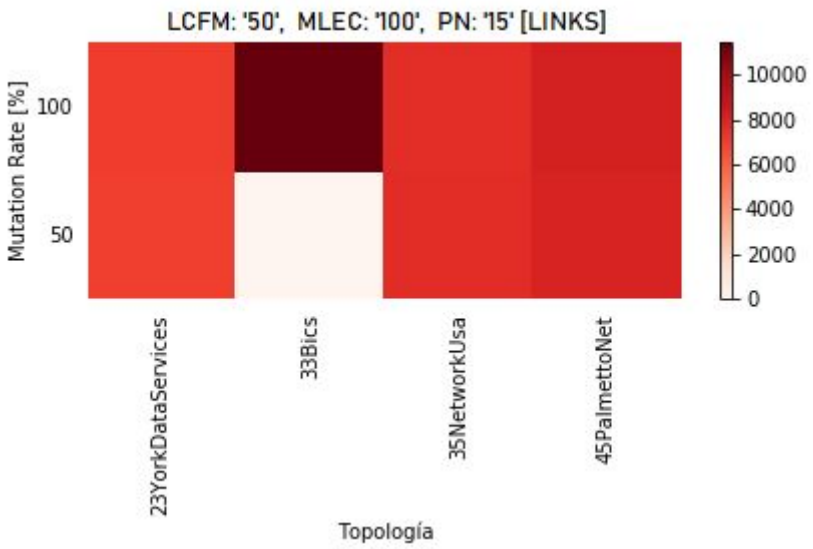


Figura 4.6. Costo de cada topología vs el TM.[Resultados al simular en python]
Para el TM se tomaron los valores(0.5 y 1), con los parámetros de LCFM:50, MLEC:100, PN:15

4.2.6. Tiempo de cómputo vs topología

Como se observa en la figura 4.7 se evidencia el tiempo máximo (café), el tiempo promedio (azul) y el tiempo mínimo (verde) que tarda el algoritmo genético en hallar la mejor solución, una iteración está compuesta por una generación que contiene 10 soluciones, en cada topología se realizaron 200 iteraciones. Por ejemplo para las topologías de PalmettoNet y Bell Canada el tiempo máximo registrado para hallar la solución óptima de la iteración fue de 100 segundos. Estos tiempos fueron obtenidos con el número de veces que se ejecutó el algoritmo que se encuentra ligado al número de posibles combinaciones de las variables (MFIL, MFIL, PN, TM) que en nuestro caso se calcula como una permutación sin conmutación, es decir 10! para Packetexchange y 12! para las demás topologías.

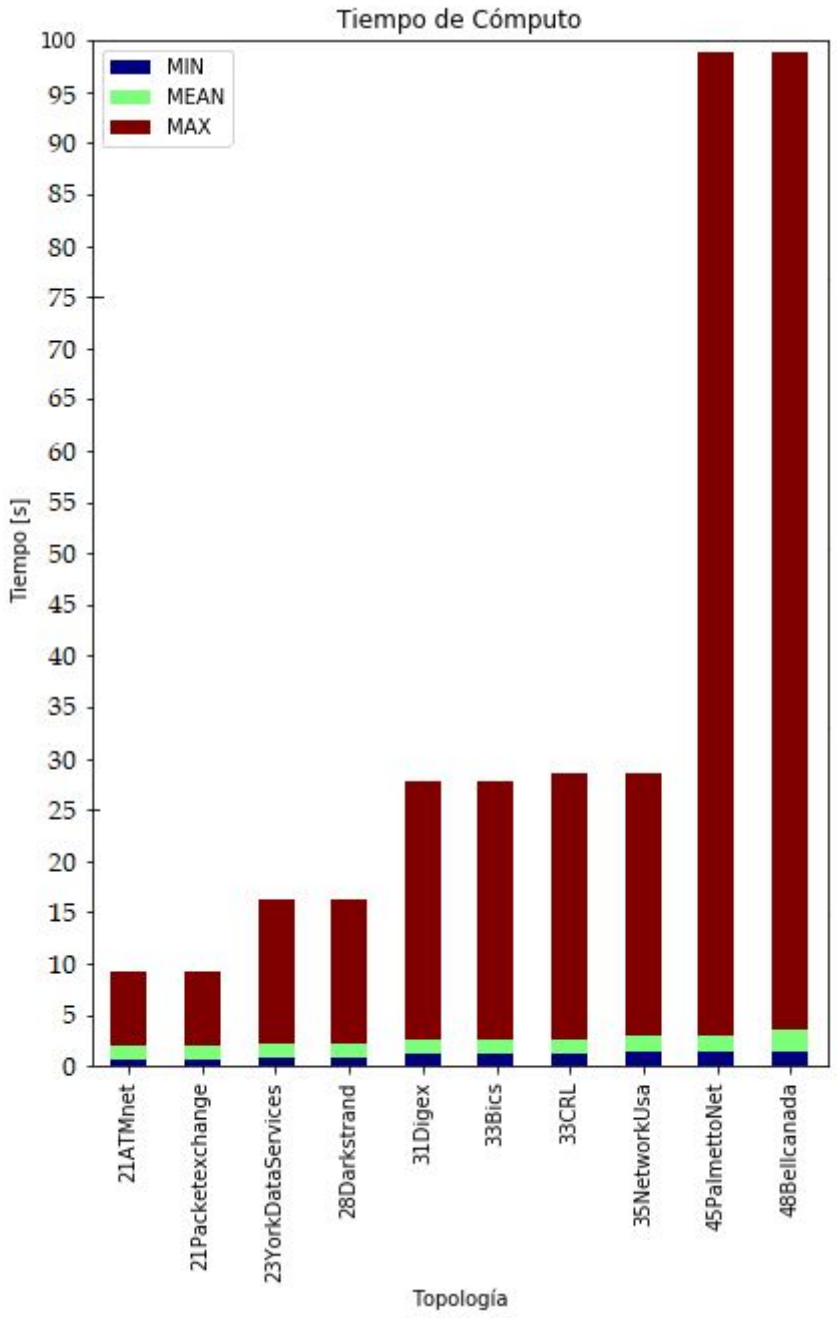


Figura 4.7. Tiempos de cómputo en las topologías de este trabajo.[Resultados al simular en python]

Para esta figura se consideraron los tiempos de cómputo máximo, mínimo y promedio.

4.3. Comparación del algoritmo genético con una búsqueda exhaustiva.

El algoritmo genético está clasificado dentro de la familia de métodos heurísticos, los cuales encuentran soluciones más rápidas pero el tamaño de muestra no explora todas las posibilidades.

El algoritmo genético inicia con una población de soluciones que son determinadas aleatoriamente por el software, lo cual significa que estadísticamente la solución inicial para el algoritmo genético desarrollado en este trabajo pueda ser la óptima o cercana a la óptima (con probabilidad de que esto suceda de 1 en cientos de millones).

Por esta razón se necesita saber cuál es la solución exacta en una topología y determinar qué tan óptimas son las soluciones que genera el algoritmo genético. Cómo determinar la solución exacta tarda mucho tiempo y no es objetivo de este trabajo pero es de referencia para medir el desempeño del algoritmo genético, se desarrolló una simulación que explora todo el universo de soluciones acotando con los siguientes parámetros fijos los cuales reducen considerablemente las soluciones. Posteriormente se corre con los mismos parámetros acotados en el algoritmo genético y se compara la solución exacta vs el desempeño del algoritmo genético.

Características de la topología evaluada para este caso:
Nombre de la Topología: AtmNet
Número de nodos: 21
Controladores por activar: 1
Proveedores de servicio: 1
Latencia de configuración de flujo máximo: 100 milisegundos
Máxima latencia entre controladores: 50 milisegundos
Porcentaje de nodo: 1
Tasa de Mutación: 1
Costo de cada enlace: 100

Al realizar la búsqueda exhaustiva sobre la topología indicada previamente, se encuentra el menor costo total por nodo, es decir para este caso la primer opción es ubicar el controlador en el nodo 12 con costo de 6200, seguido del controlador en el

nodo 11 con valor de 6800 y finalmente ubicar el controlador en el nodo 6 con costo 7100, como se observa en la tabla 4.3

Costo total	Ubicando el controlador en
6200	12
6800	11
7100	6
7200	2
7300	20
8000	3
8200	19
8300	7
8500	0
8700	18
9300	1
9400	4
9400	5
9400	17
9800	9
9800	14
10100	10
10300	16
11000	8
11600	13

Tabla 4.3. Ranking de costos en una evaluación exhaustiva.

Como se observa en la figura 4.8. se resaltan en rojo las tres mejores ubicaciones del controlador en cada nodo realizando una búsqueda exhaustiva.

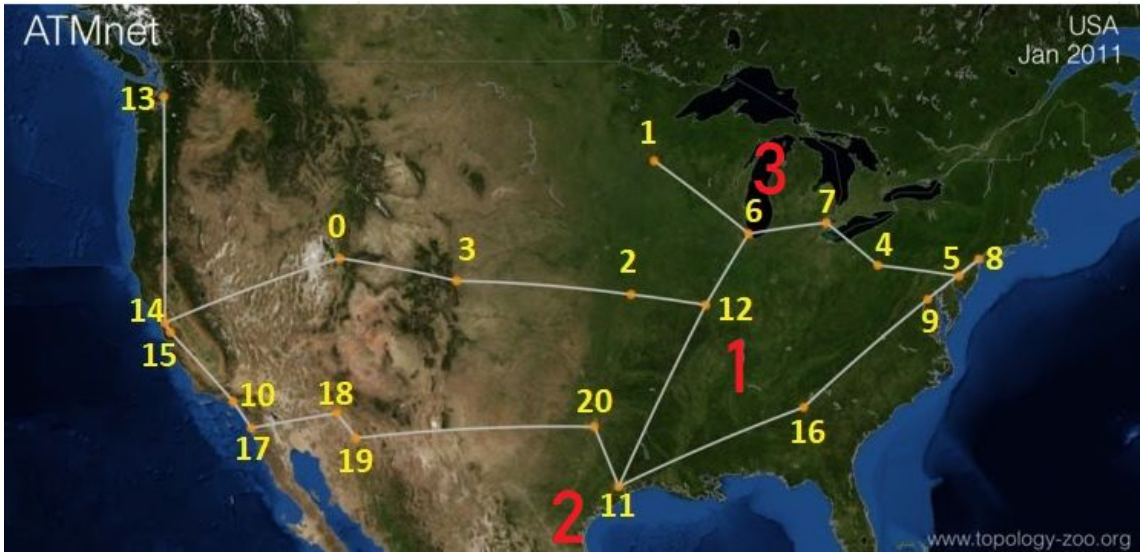


Figura 4.8. Top 3 mejores ubicaciones del controlador con método exhaustivo. [Gráfica tomada del Topology Zoo y modificada personalmente]

Ahora se toman los resultados producidos por el algoritmo genético que genera una población de soluciones, para este caso se toma una población aleatoria inicial de 3 soluciones aleatorias como se ve en la tabla 4.4. las cuales tendrán una mutación y selección de una solución, es decir el 33,33% de la población. Luego de realizar 200 iteraciones la población de soluciones que genera el algoritmo se ve en la tabla 4.5, las tres soluciones finales se observan en la figura 4.9 resaltadas en números rojos.:

Costo total	Ubicando el controlador en
10300	16
8300	7
8500	0

Tabla 4.4. Población inicial generada aleatoriamente por el algoritmo genético.

Costo total	Ubicando el controlador en
6200	12
7100	6
7300	20

Tabla 4.5. Población final generada por el algoritmo genético.

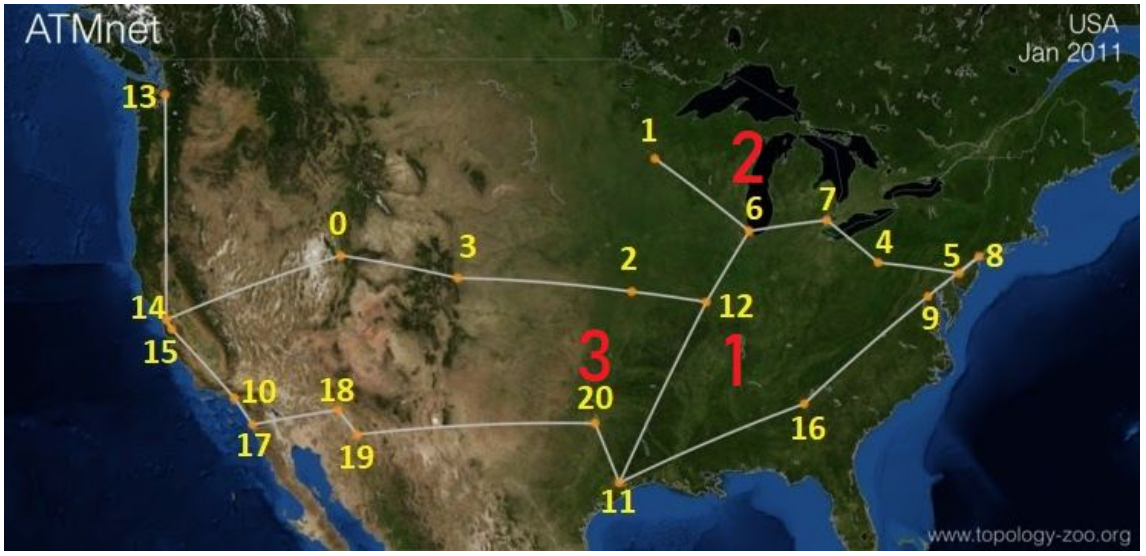


Figura 4.9. Top 3 mejores ubicaciones del controlador con algoritmo genético. [Gráfica tomada del Topology Zoo y modificada personalmente]

De acuerdo con lo anterior se puede observar que el costo total de las 3 soluciones óptimas de la búsqueda exhaustiva dan un total de 20.100, mientras que las tres soluciones óptimas generadas por el algoritmo genético suman 20.600. Se debe tener considerar este dato puesto que si se ejecutara de nuevo el algoritmo genético el resultado puede variar debido a que el proceso de mutación es aleatorio en el algoritmo genético. Se puede concluir con esta gráfica que el algoritmo siempre buscará la mejor solución porque el mecanismo de selección elimina las peores soluciones, en este caso el 33,33%

De las 10 topologías consideradas en este trabajo se evidencia las características de cada una de ellas, por lo tanto no todos los parámetros satisfacen a las topologías evaluadas ya que estos parámetros no cumplen con las restricciones planteadas, en consecuencia algunos casos sí encontrarán soluciones (cuando cumplen con las restricciones) y en otros casos no (cuando no cumplen con las restricciones).

5. CONCLUSIONES

A continuación se presentan las conclusiones de los resultados del capítulo anterior y del algoritmo genético

Resultados

- Para la topología Packetexchange se consideraron los valores de 275ms y 300ms. Para la topología de Darkstrand se tomaron los valores de 200ms, 225ms y 250ms y finalmente para Bellcanada se tomó 200ms, 225ms y 250ms. Esto debido a las distancias que hay entre nodos, lo cual indica que a mayor distancia entre nodos el LCFM y el MLEC aumenta debido a que el desplazamiento de un nodo a otro toma más tiempo, si se aplicarán valores de LCFM y MLEC inferiores a los mostrados en esta conclusión el algoritmo no encontraría soluciones ya que las restricciones serían muy exigentes y no se encontraría optimización.

- La primer generación de muestra se crea de forma aleatoria, luego pasa por el algoritmo genético, como resultado se obtiene una optimización de costo, el porcentaje de optimización varía acorde a cada topología y a los parámetros de entrada (LCFM, MLEC, PN, TM), se puede evidenciar en las gráficas de “costo de la topología antes y después de la optimización” (ver anexo), en este tipo de gráficas se observa optimización de costos entre el rango del 2% al 8% respecto al original.

- La latencia de configuración de flujo máximo varía acorde con cada topología, esto debido a la distancia entre nodos, por lo tanto cuando la distancia entre nodos es mayor, la latencia de configuración de flujo aumenta.

- El tasa de mutación(TM) como variable de ingreso al 50% ó 100% no hace que el costo final de la solución tenga una diferencia notable, ya que al 50% y al 100% el costo final de la solución es similar.

- Las topologías probadas que cuentan con mayor número de nodos (PalmettoNet con 45 y Bellcanada con 48) fueron las que mayor tiempo de cómputo tomaron para llegar a una solución (100 segundos), mientras que las topologías con menor cantidad de nodos (ATMnet y Packetexchange con 21 nodos) tomaron 8 segundos en encontrar una solución, lo cual indica que el tiempo de cómputo incrementa con la cantidad de nodos.

- El tiempo de cómputo máximo de cada solución es directamente proporcional a la cantidad de nodos que tiene la topología, se evidencia que para las topologías ATM NEt y PacketExchange que contienen 21 nodos, el tiempo máximo de computo no supera los 10 segundos, mientras que para las topologías de PalmettoNet y BellCananda que contienen 45 y 48 nodos respectivamente, el tiempo de cómputo por cada solución puede tomar hasta 100 segundos.

- Las principales variables que permiten la ejecución del algoritmo genético son la LCFM (Latencia de configuración de flujo máximo) y MLEC (Máxima latencia entre controladores), debido a que estas variables contienen restricciones de tiempo. Por esta razón no todas las topologías pueden ser ejecutadas con los mismos valores de LCFM y MLEC.

- Se realizaron 200 iteraciones por cada simulación, esto acorde con los resultados, se puede observar que en la iteración 175 el algoritmo estabiliza las soluciones al optimizar el costo.

Conclusiones del algoritmo genético

- El algoritmo genético tardaba en encontrar soluciones en el rango de 1 a 100 segundos, esto gracias a un parámetro de tiempo límite que hace parte del algoritmo, de no ser así, el tiempo de cálculo de una solución pudo ser infinito, debido a que no hubiese forma de encontrar solución. De esta manera, el tiempo de ejecución de las 10 topologías estudiadas fue de 173 horas. Este tiempo varía de acuerdo con la máquina que se ejecute el algoritmo, para este caso se utilizó un dispositivo Intel® Core™ i7-8750H Processor 2.2 GHz con

32 Gb de RAM. Este tiempo de procesamiento puede mejorar con una máquina que contenga capacidad superior.

- El número de nodos afecta directamente el tiempo de respuesta de solución del algoritmo, cuando se tenía la topología ATM Net de 21 nodos el tiempo de ejecución del algoritmo fue de 4 horas, mientras que con la topología Bell Canada de 48 nodos el tiempo de ejecución del algoritmo fue de 32 horas.

Conclusiones por objetivos

1. Se propuso y evaluó un algoritmo genético para redes WAN adquiridas del Topology Zoo donde por medio de una función objetivo de costo se halló la mejor ubicación del controlador.

2. Se realizó una investigación intensiva sobre el estado del arte actual de los métodos heurísticos que hablan sobre el problema de la ubicación del controlador en redes WAN, evidenciando que uno de los principales criterios de optimización son multiobjetivo y las restricciones están basadas en latencia en su mayoría, las técnicas más utilizadas fueron crear algoritmos heurísticos con el fin de llegar a una pronta solución sin recorrer todo el universo de soluciones posibles.

3. Se propone un algoritmo genético en el cual se aborda el problema de ubicación del controlador optimizando los costos a nivel económico, seguido de restricciones de latencia entre controladores y nodos, se utilizó un algoritmo genético con operadores de mutación y selección, las topologías utilizadas en este trabajo fueron tomadas del Topology Zoo, las cuales son topologías reales de proveedores de servicios de Telecomunicaciones.

4. El algoritmo genético de este trabajo se implementa en el software python 3.7 gracias a que es un lenguaje de programación de rápida ejecución y que permite realizar gráficas con facilidad sin necesidad de integrar otra plataforma de desarrollo.

5. Al validar el algoritmo genético se observa que no todos los parámetros satisfacen las restricciones en el momento de evaluar cada caso, esto debido a la naturaleza de cada topología la cual contiene un número diferente de nodos y distancias variables entre nodo y nodo, lo cual hace que los parámetros no generen resultados en todos los

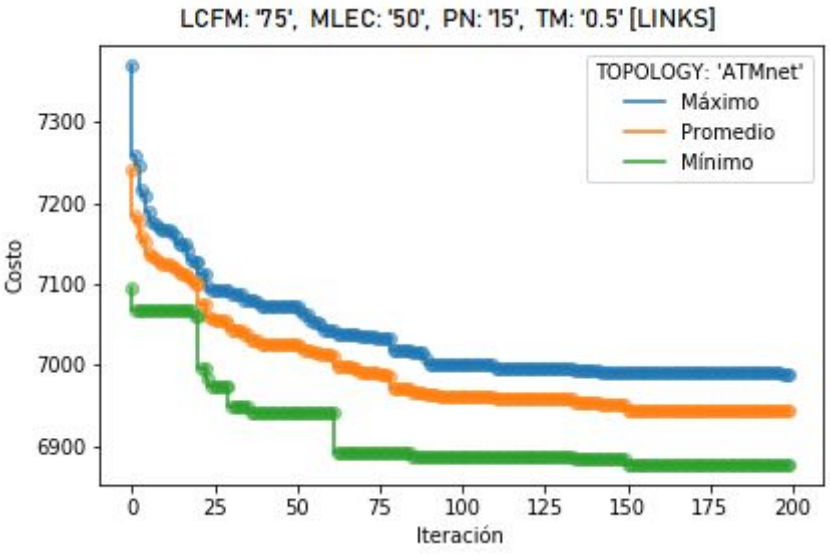
casos. El desempeño varía de acuerdo a la cantidad de nodos, entre mayor número de nodos el tiempo de ejecución del algoritmo aumenta.

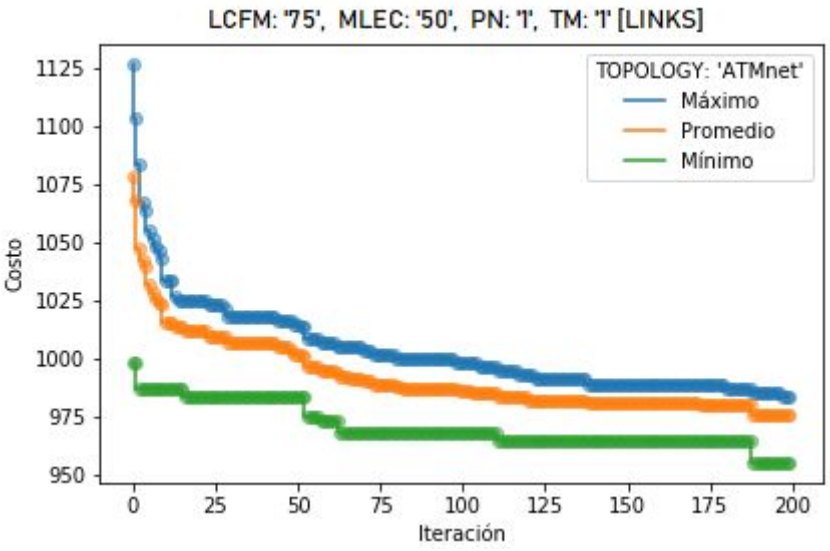
Anexo A

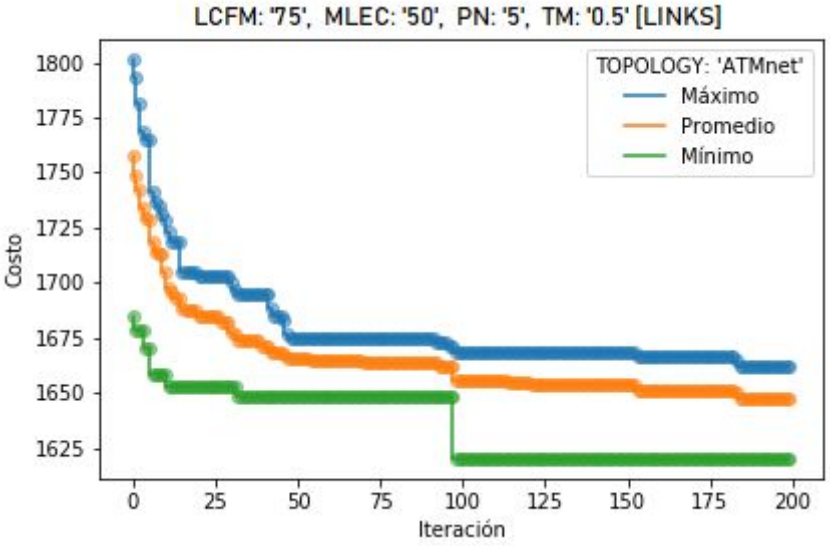
A continuación se muestran los resultados considerando las siguientes variables LCFM (Latencia de configuración de flujo máximo), MLEC (Máxima latencia entre controladores), PN(Porcentaje de nodo), TM(tasa de mutación). Las siguientes gráficas son obtenidas al simular el algoritmo en python.

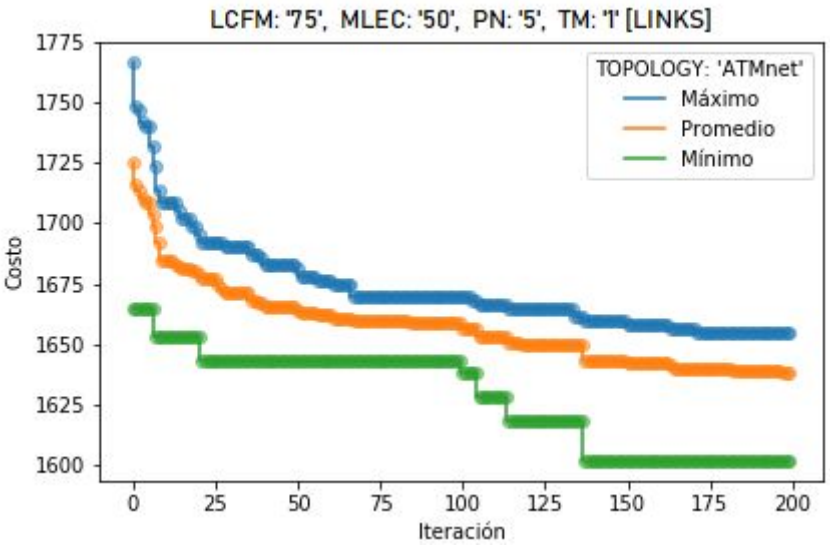
- **Las siguientes gráficas muestran el costo de la topología a lo largo de cada iteración**

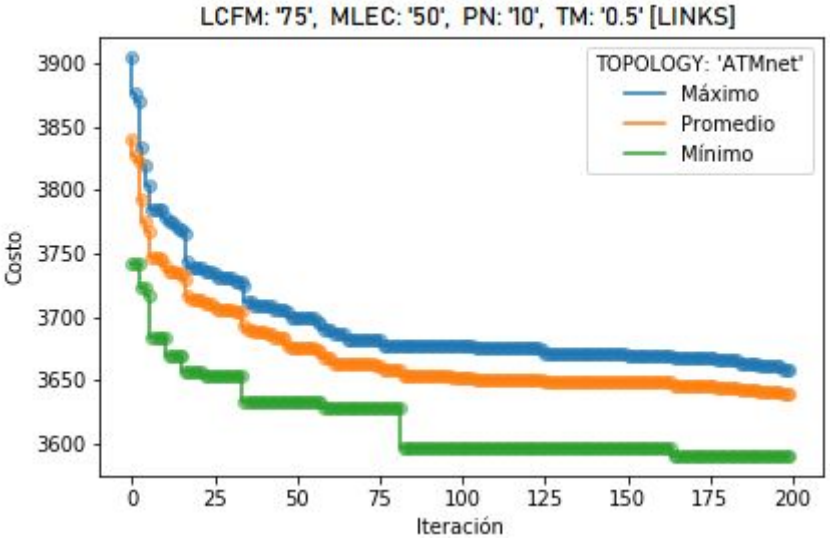
Para la topología ATM NET

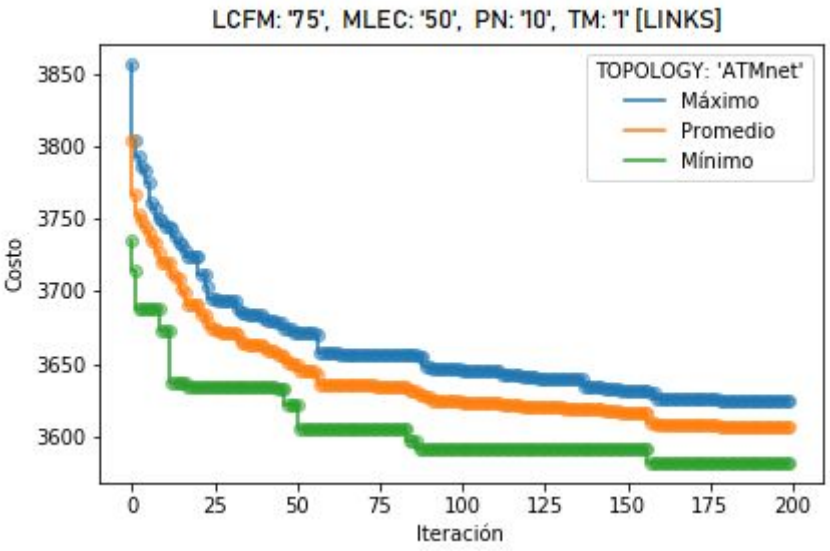


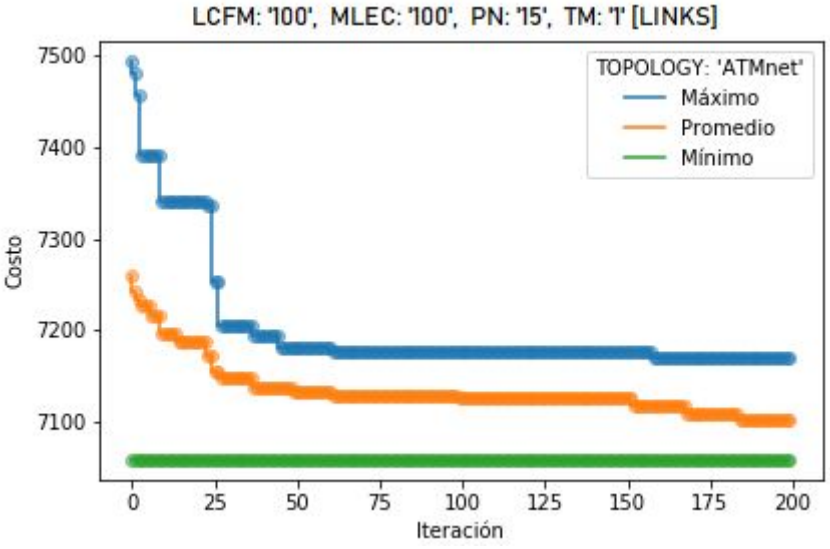

LCFM: '75', MLEC: '50', PN: '1', TM: '1' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
1125
1100
1075
1050
1025
1000
975
950
0
25
50
75
100
125
150
175
200

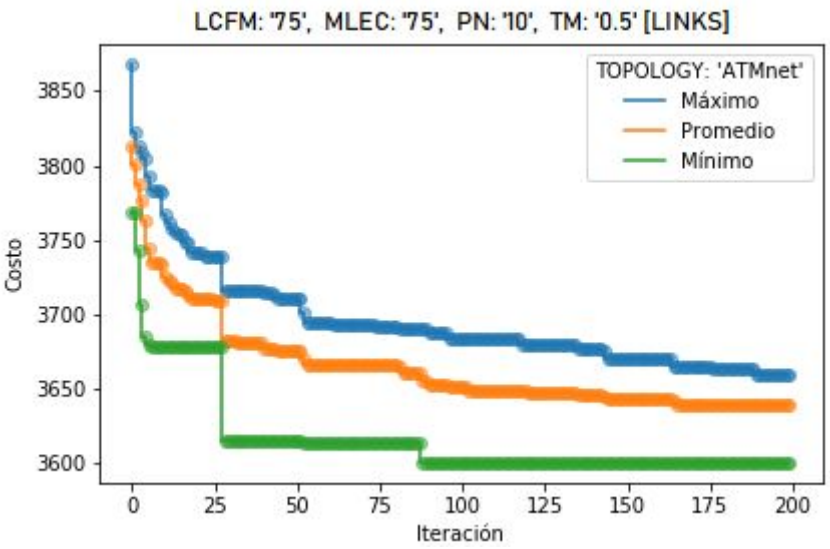

LCFM: '75', MLEC: '50', PN: '5', TM: '0.5' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
1800
1775
1750
1725
1700
1675
1650
1625
0
25
50
75
100
125
150
175
200

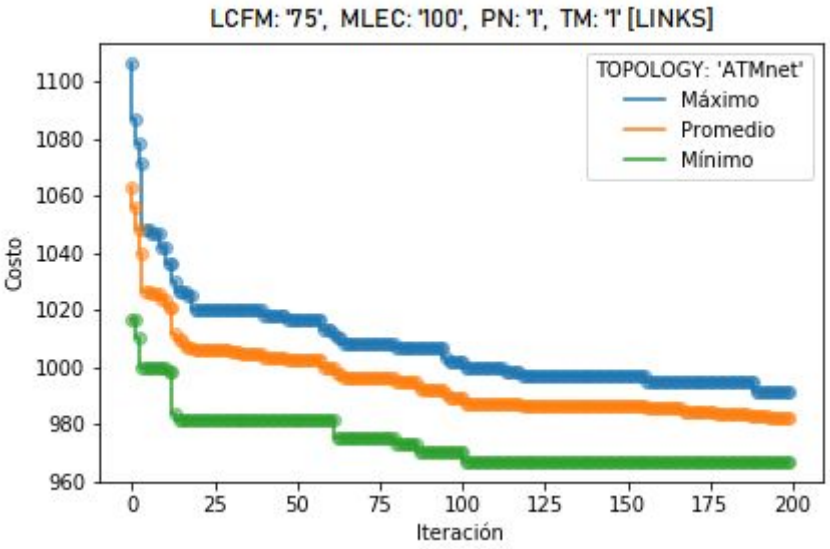

LCFM: '75', MLEC: '50', PN: '5', TM: '1' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
1775
1750
1725
1700
1675
1650
1625
1600
0
25
50
75
100
125
150
175
200

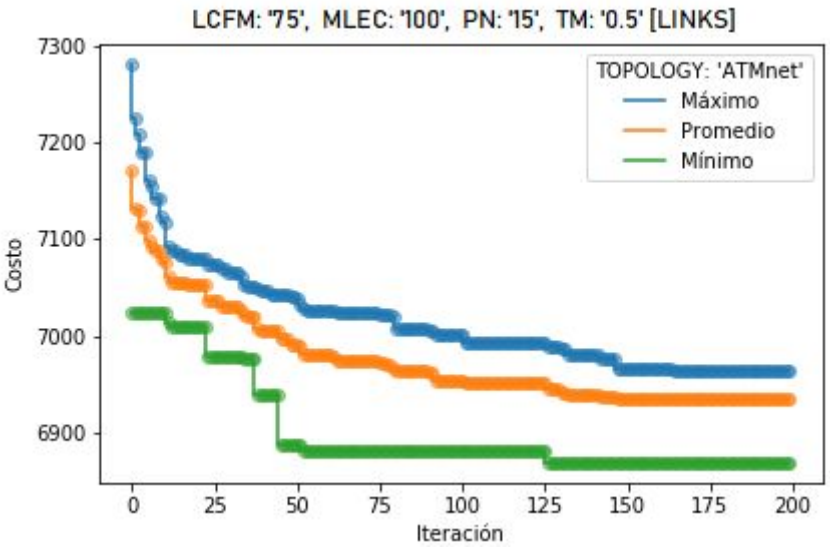

LCFM: '75', MLEC: '50', PN: '10', TM: '0.5' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
3900
3850
3800
3750
3700
3650
3600
0
25
50
75
100
125
150
175
200

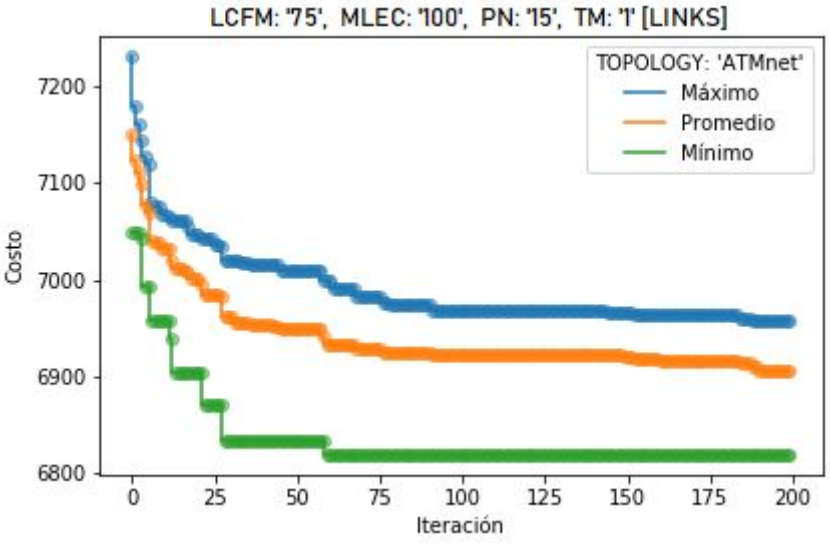

LCFM: '75', MLEC: '50', PN: '10', TM: 'l' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
3850
3800
3750
3700
3650
3600
0
25
50
75
100
125
150
175
200

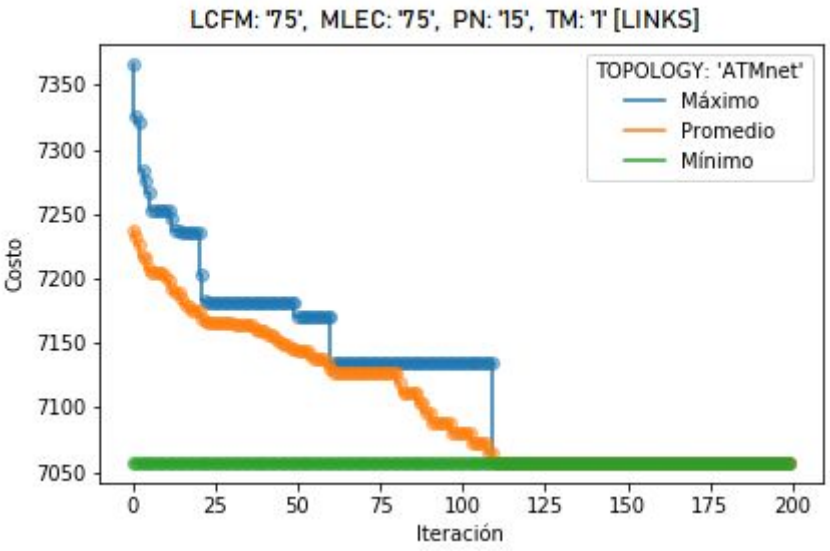

LCFM: '100', MLEC: '100', PN: '15', TM: 'l' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
7500
7400
7300
7200
7100
0
25
50
75
100
125
150
175
200

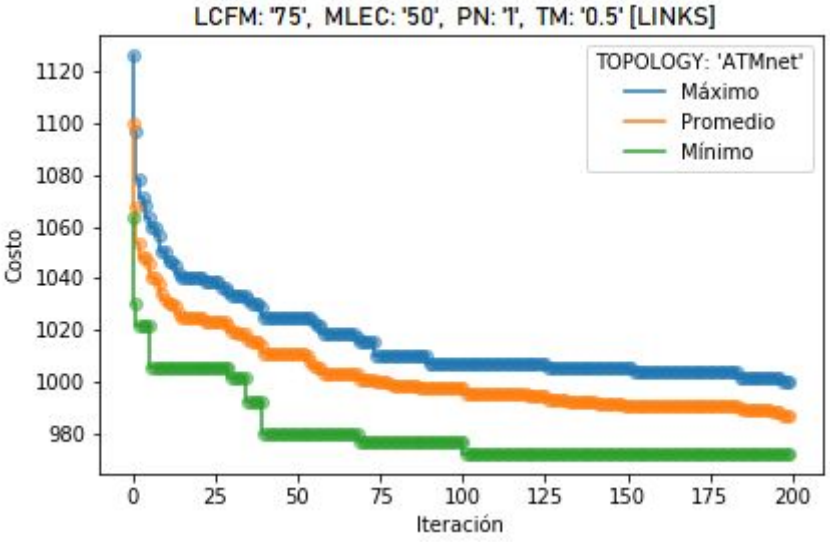

LCFM: '75', MLEC: '75', PN: '10', TM: '0.5' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
3850
3800
3750
3700
3650
3600
0
25
50
75
100
125
150
175
200


LCFM: '75', MLEC: '100', PN: '1', TM: '1' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
1100
1080
1060
1040
1020
1000
980
960
0
25
50
75
100
125
150
175
200

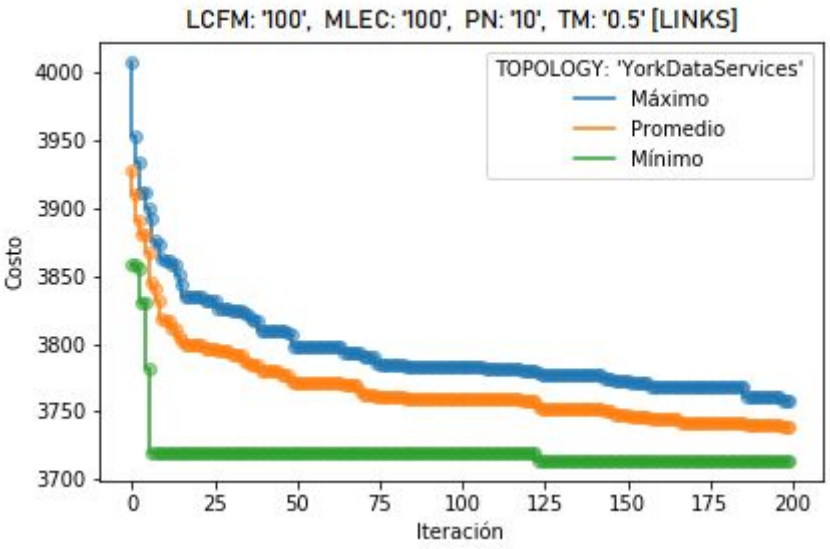

LCFM: '75', MLEC: '100', PN: '15', TM: '0.5' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
7300
7200
7100
7000
6900
0
25
50
75
100
125
150
175
200

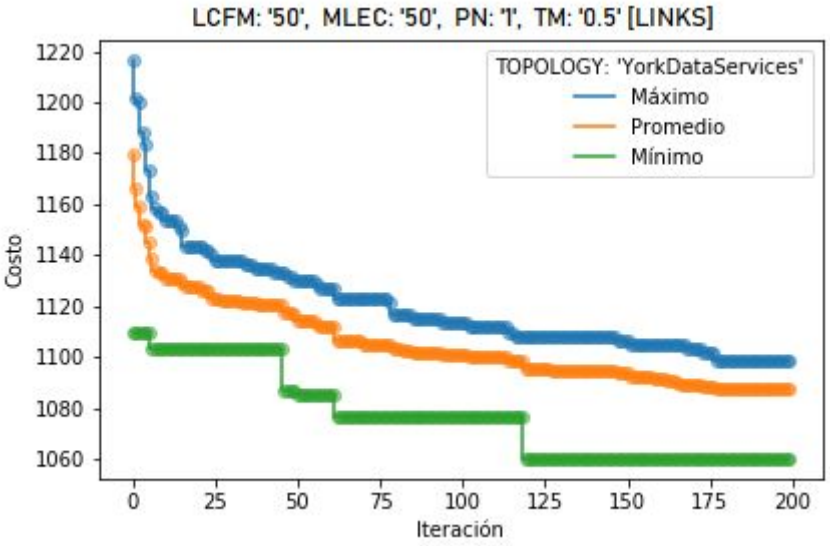

LCFM: '75', MLEC: '100', PN: '15', TM: '1' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
7200
7100
7000
6900
6800
0
25
50
75
100
125
150
175
200

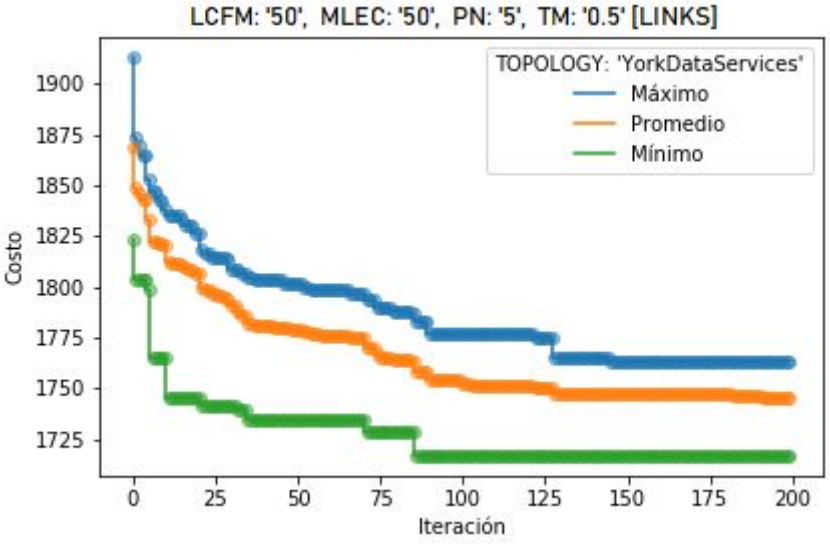

LCFM: '75', MLEC: '75', PN: '15', TM: 'I' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
7350
7300
7250
7200
7150
7100
7050
0
25
50
75
100
125
150
175
200

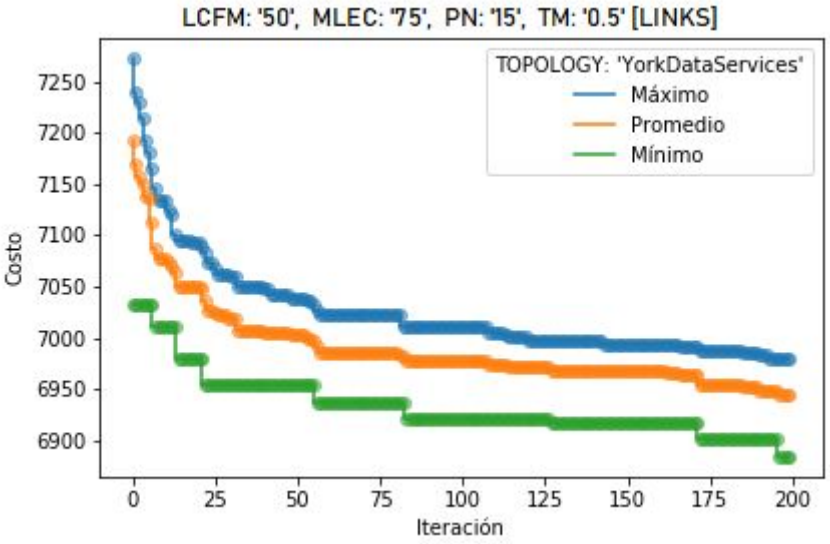

LCFM: '75', MLEC: '50', PN: 'I', TM: '0.5' [LINKS]
TOPOLOGY: 'ATMnet'
Máximo
Promedio
Mínimo
Costo
Iteración
1120
1100
1080
1060
1040
1020
1000
980
0
25
50
75
100
125
150
175
200

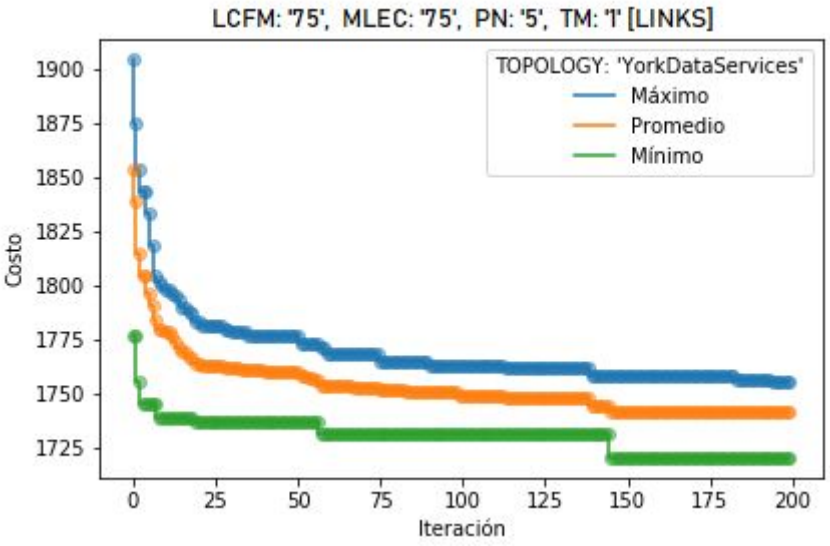
Para la topología York Data Services

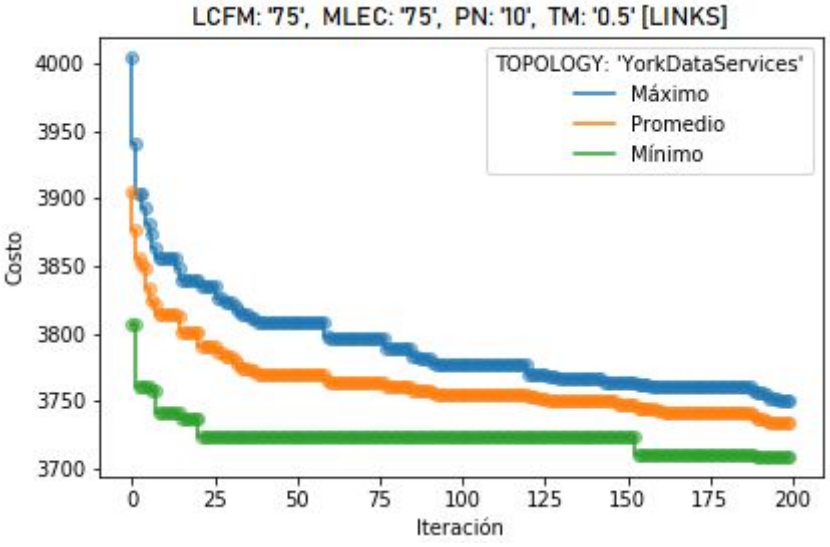


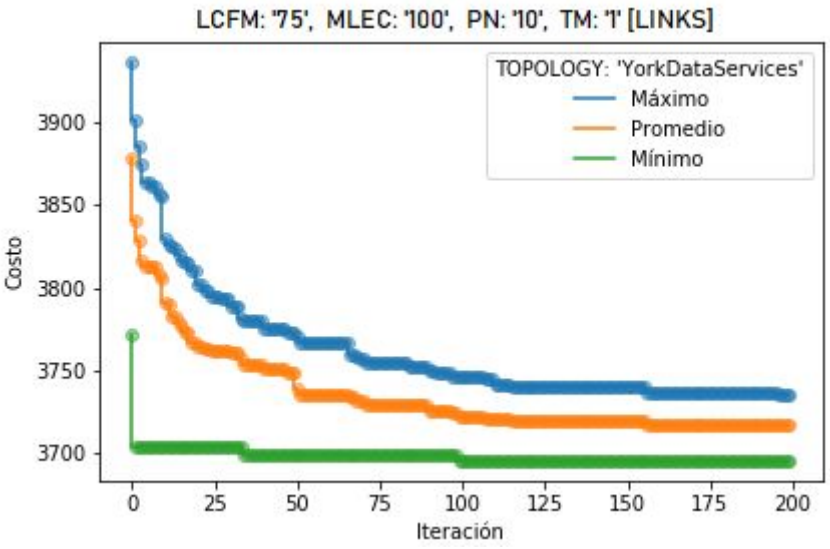

LCFM: '50', MLEC: '50', PN: '1', TM: '0.5' [LINKS]
TOPOLOGY: 'YorkDataServices'
Máximo
Promedio
Mínimo
Costo
Iteración
1220
1200
1180
1160
1140
1120
1100
1080
1060
0
25
50
75
100
125
150
175
200

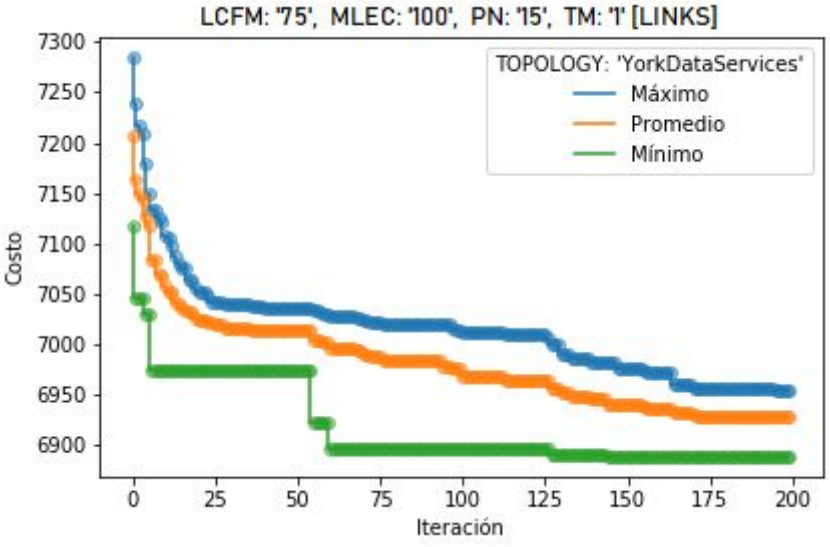

LCFM: '50', MLEC: '50', PN: '5', TM: '0.5' [LINKS]
TOPOLOGY: 'YorkDataServices'
Máximo
Promedio
Mínimo
Costo
Iteración
1900
1875
1850
1825
1800
1775
1750
1725
0
25
50
75
100
125
150
175
200

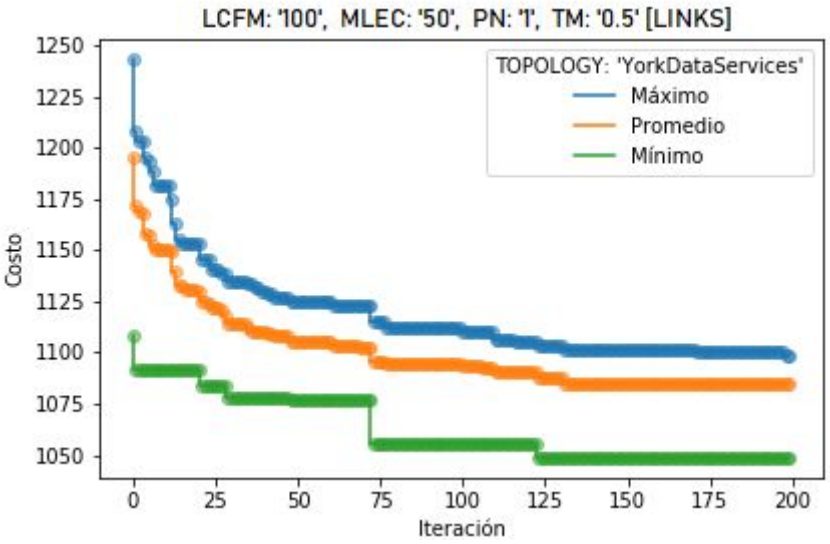

LCFM: '50', MLEC: '75', PN: '15', TM: '0.5' [LINKS]
TOPOLOGY: 'YorkDataServices'
Máximo
Promedio
Mínimo
Costo
7250
7200
7150
7100
7050
7000
6950
6900
0
25
50
75
100
125
150
175
200
Iteración


LCFM: '75', MLEC: '75', PN: '5', TM: '1' [LINKS]
TOPOLOGY: 'YorkDataServices'
Máximo
Promedio
Mínimo
Costo
Iteración
1900
1875
1850
1825
1800
1775
1750
1725
0
25
50
75
100
125
150
175
200

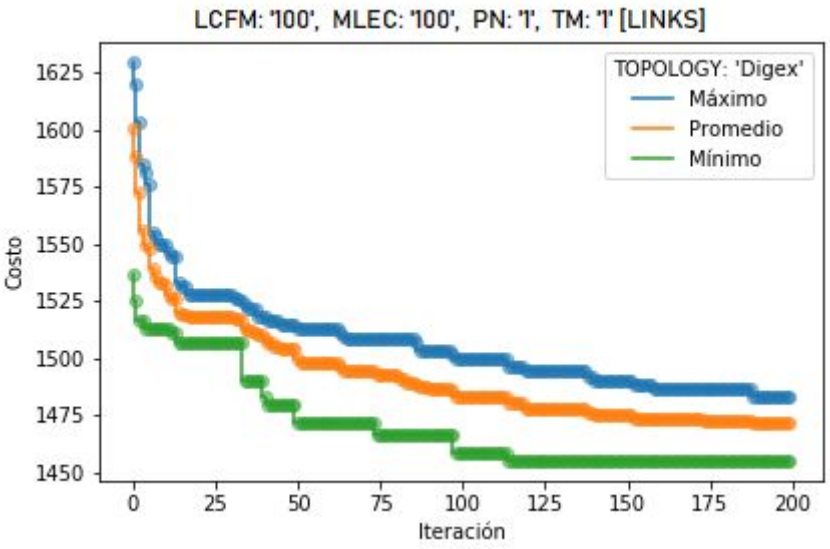

LCFM: '75', MLEC: '75', PN: '10', TM: '0.5' [LINKS]
TOPOLOGY: 'YorkDataServices'
Máximo
Promedio
Mínimo
Costo
Iteración
4000
3950
3900
3850
3800
3750
3700
0
25
50
75
100
125
150
175
200

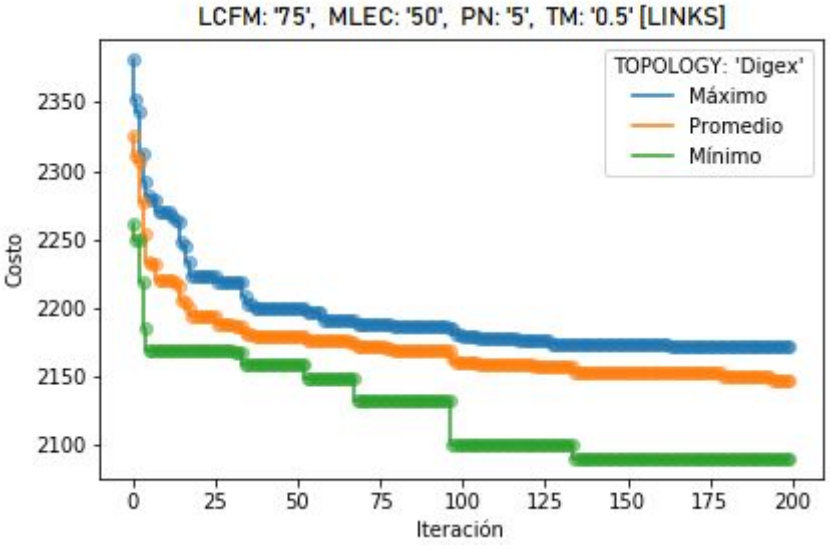

LCFM: '75', MLEC: '100', PN: '10', TM: 'l' [LINKS]
TOPOLOGY: 'YorkDataServices'
Máximo
Promedio
Mínimo
Costo
3900
3850
3800
3750
3700
0
25
50
75
100
125
150
175
200
Iteración

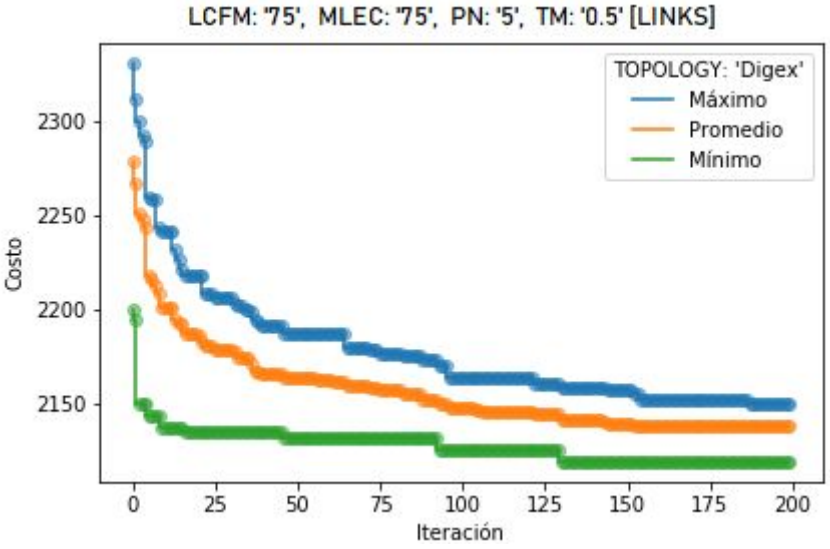

LCFM: '75', MLEC: '100', PN: '15', TM: 'l' [LINKS]
TOPOLOGY: 'YorkDataServices'
Máximo
Promedio
Mínimo
Costo
7300
7250
7200
7150
7100
7050
7000
6950
6900
0
25
50
75
100
125
150
175
200
Iteración

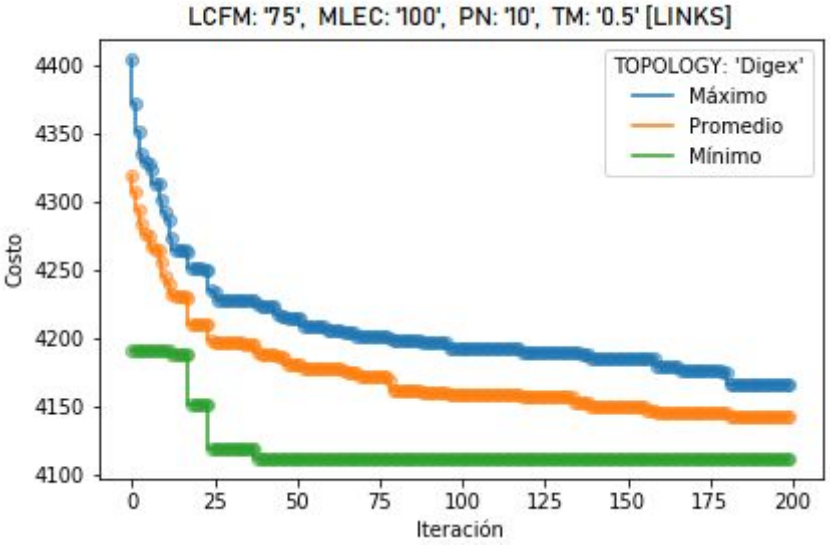


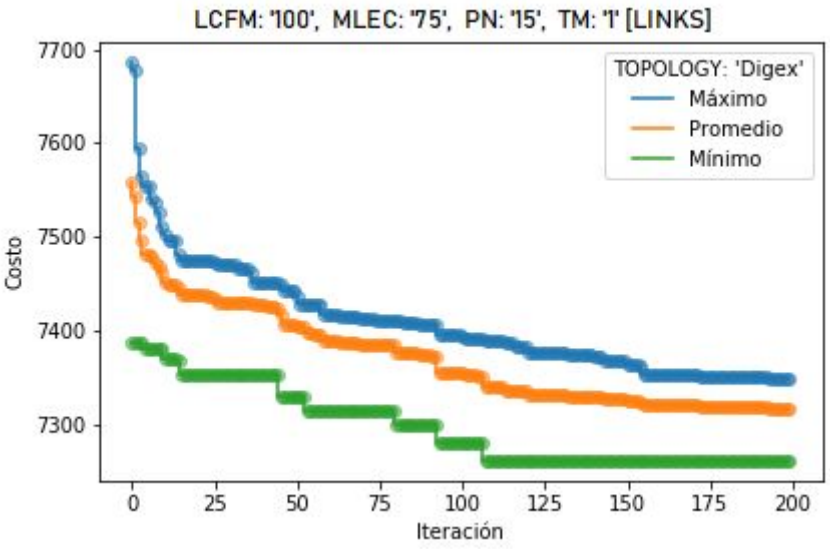
Para la topología Digex


LCFM: '100', MLEC: '100', PN: '1', TM: '1' [LINKS]
TOPOLOGY: 'Digex'
Máximo
Promedio
Mínimo
Costo
Iteración
1625
1600
1575
1550
1525
1500
1475
1450
0
25
50
75
100
125
150
175
200

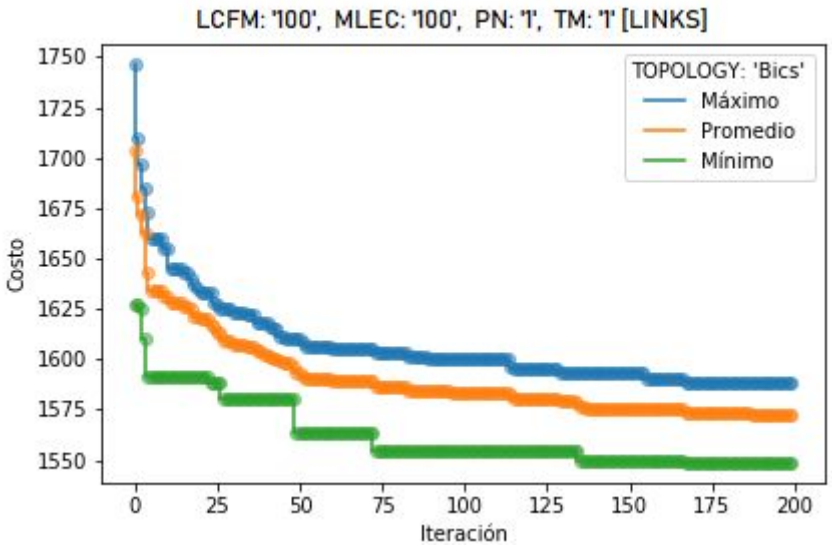

LCFM: '75', MLEC: '50', PN: '5', TM: '0.5' [LINKS]
TOPOLOGY: 'Digex'
Máximo
Promedio
Mínimo
Costo
Iteración
2350
2300
2250
2200
2150
2100
0
25
50
75
100
125
150
175
200

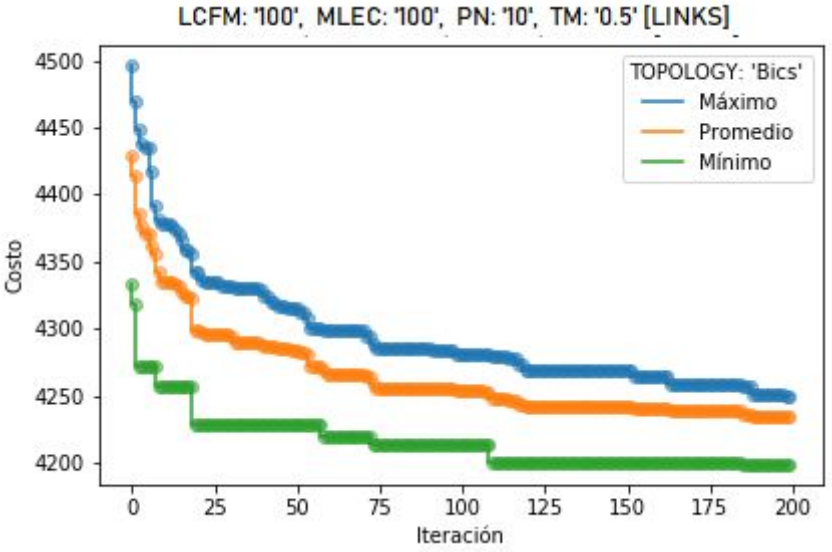

LCFM: '75', MLEC: '75', PN: '5', TM: '0.5' [LINKS]
TOPOLOGY: 'Digex'
Máximo
Promedio
Mínimo
Costo
Iteración
2300
2250
2200
2150
0
25
50
75
100
125
150
175
200

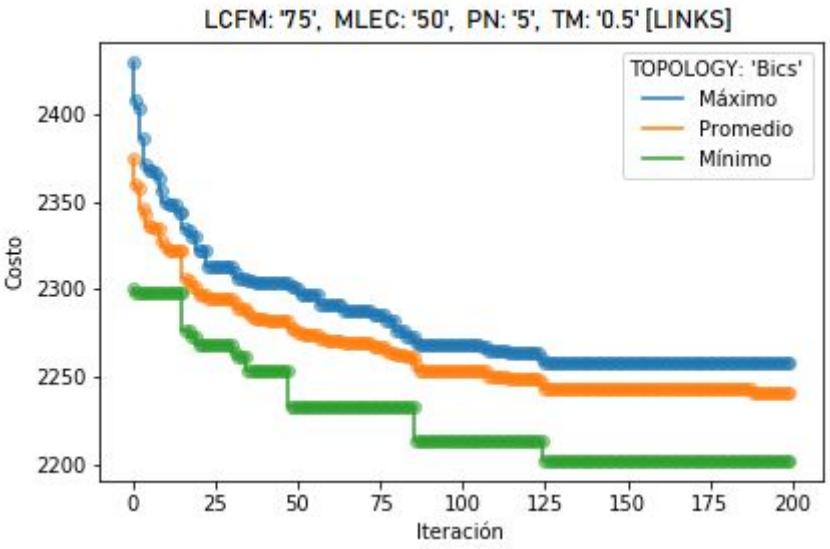

LCFM: '75', MLEC: '100', PN: '10', TM: '0.5' [LINKS]
TOPOLOGY: 'Digex'
Máximo
Promedio
Mínimo
Costo
Iteración
4400
4350
4300
4250
4200
4150
4100
0
25
50
75
100
125
150
175
200

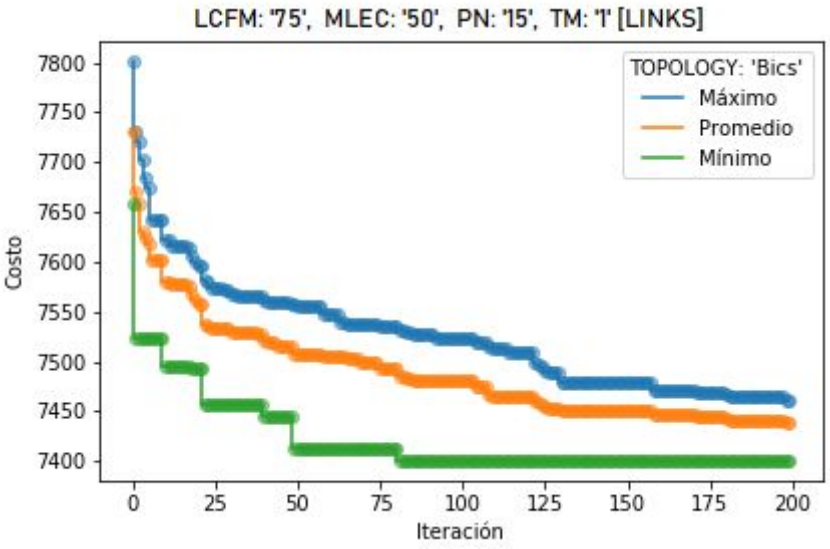


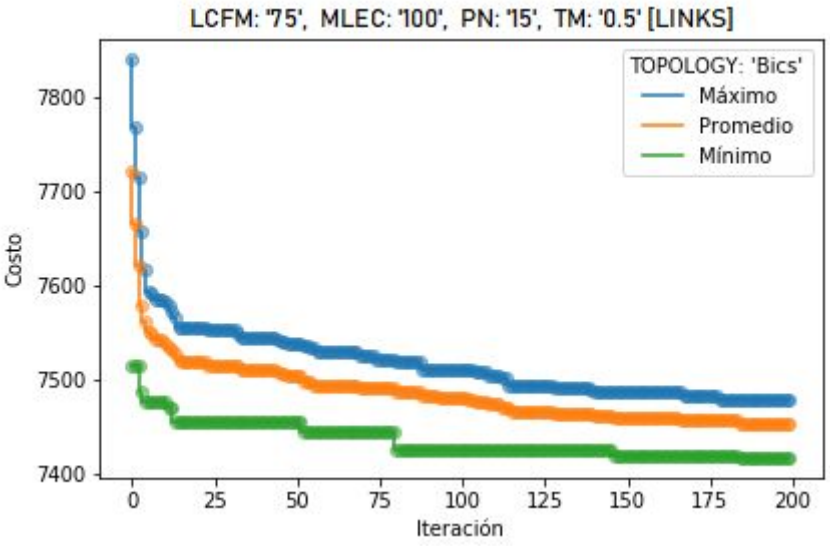
Para la topología Bics


LCFM: '100', MLEC: '100', PN: '1', TM: '1' [LINKS]
TOPOLOGY: 'Bics'
Máximo
Promedio
Mínimo
Costo
Iteración
1750
1725
1700
1675
1650
1625
1600
1575
1550
0
25
50
75
100
125
150
175
200

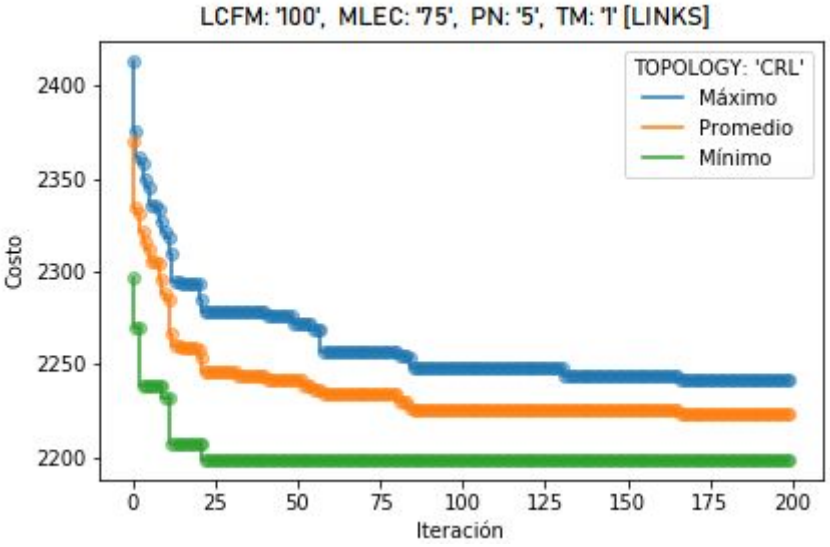

LCFM: '100', MLEC: '100', PN: '10', TM: '0.5' [LINKS]
TOPOLOGY: 'Bics'
Máximo
Promedio
Mínimo
Costo
Iteración
4500
4450
4400
4350
4300
4250
4200
0
25
50
75
100
125
150
175
200

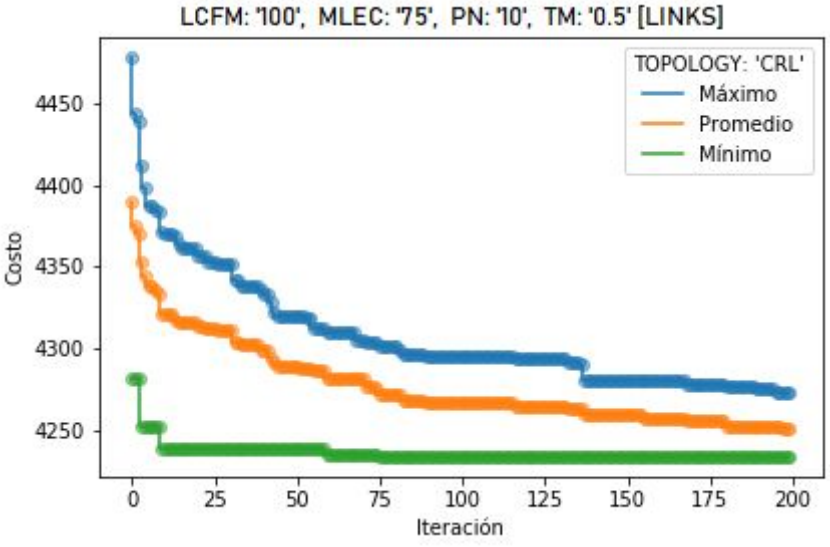

LCFM: '75', MLEC: '50', PN: '5', TM: '0.5' [LINKS]
TOPOLOGY: 'Bics'
Máximo
Promedio
Mínimo
Costo
Iteración
2400
2350
2300
2250
2200
0
25
50
75
100
125
150
175
200

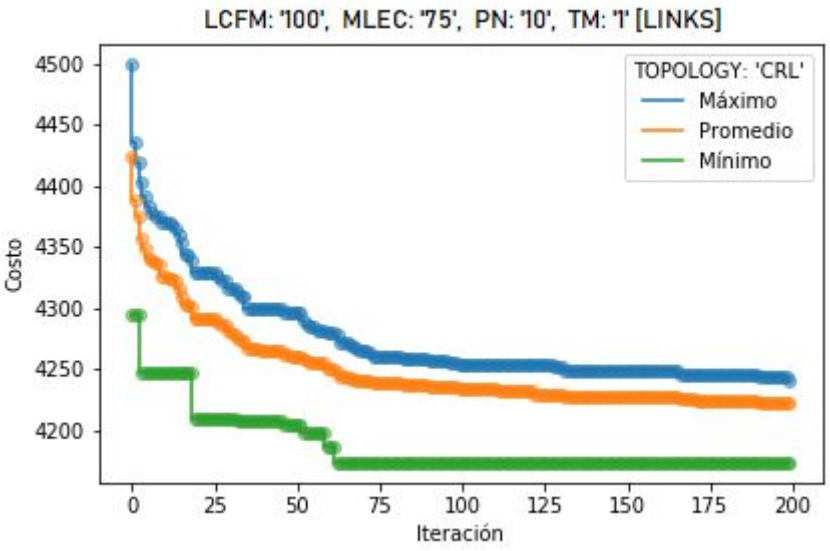

LCFM: '75', MLEC: '50', PN: '15', TM: '1' [LINKS]
TOPOLOGY: 'Bics'
Máximo
Promedio
Mínimo
Costo
Iteración
7800
7750
7700
7650
7600
7550
7500
7450
7400
0
25
50
75
100
125
150
175
200

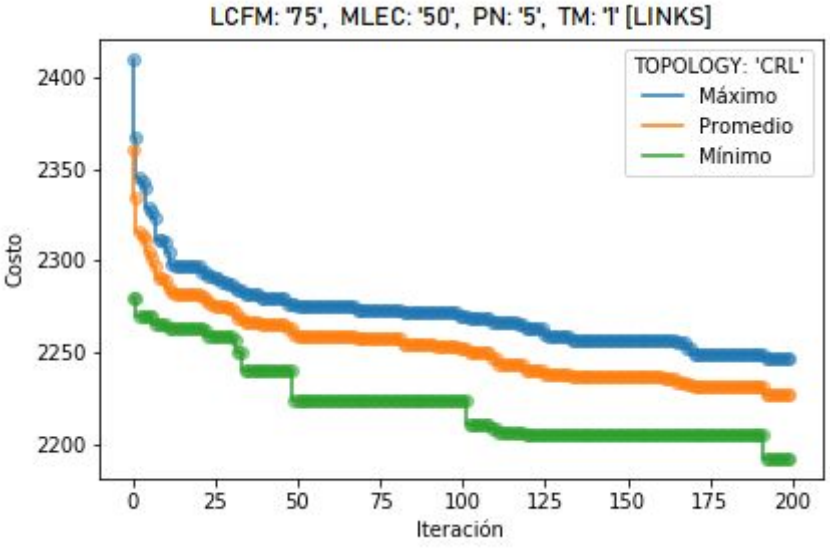


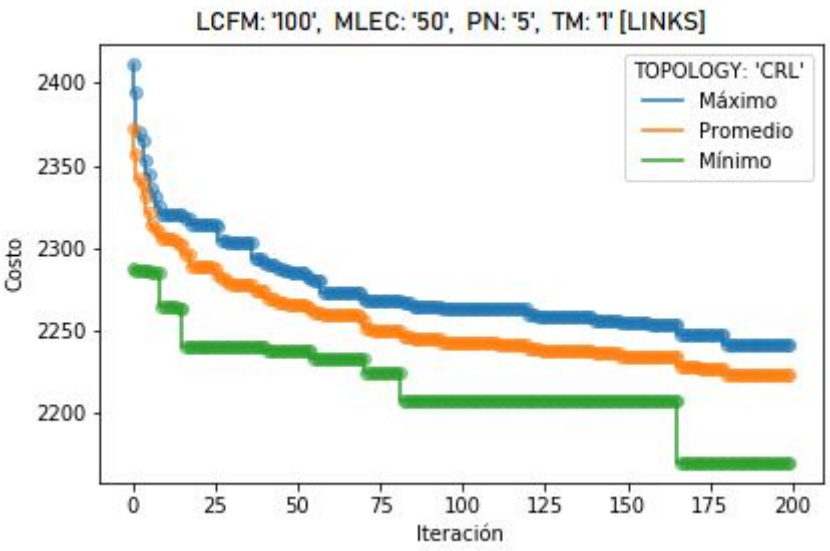
Para la topología CRL


LCFM: '100', MLEC: '75', PN: '5', TM: '1' [LINKS]
TOPOLOGY: 'CRL'
Máximo
Promedio
Mínimo
Costo
Iteración
2400
2350
2300
2250
2200
0
25
50
75
100
125
150
175
200

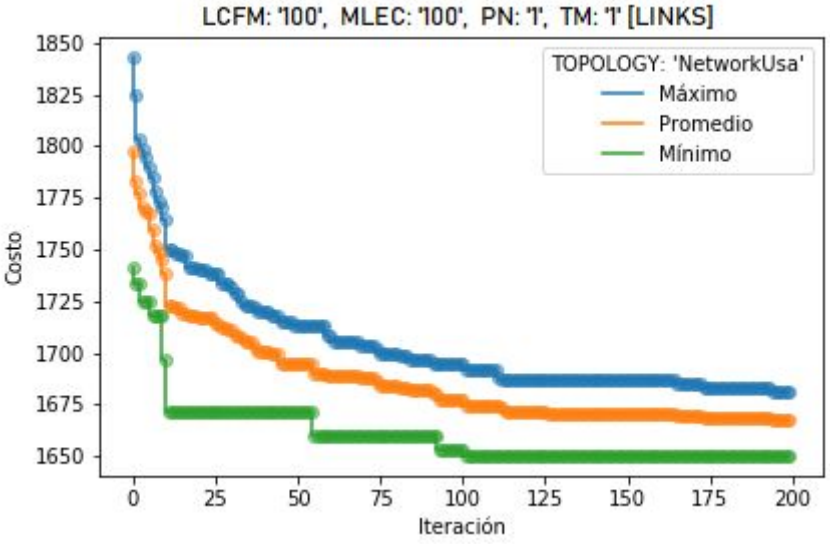

LCFM: '100', MLEC: '75', PN: '10', TM: '0.5' [LINKS]
TOPOLOGY: 'CRL'
Máximo
Promedio
Mínimo
Costo
Iteración
4450
4400
4350
4300
4250
0
25
50
75
100
125
150
175
200

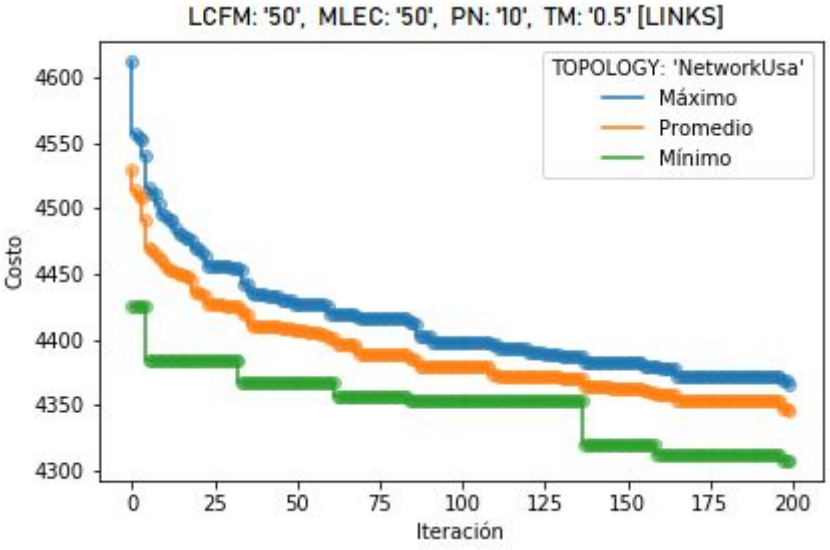

LCFM: '100', MLEC: '75', PN: '10', TM: 'l' [LINKS]
TOPOLOGY: 'CRL'
Máximo
Promedio
Mínimo
Costo
Iteración
4500
4450
4400
4350
4300
4250
4200
0
25
50
75
100
125
150
175
200

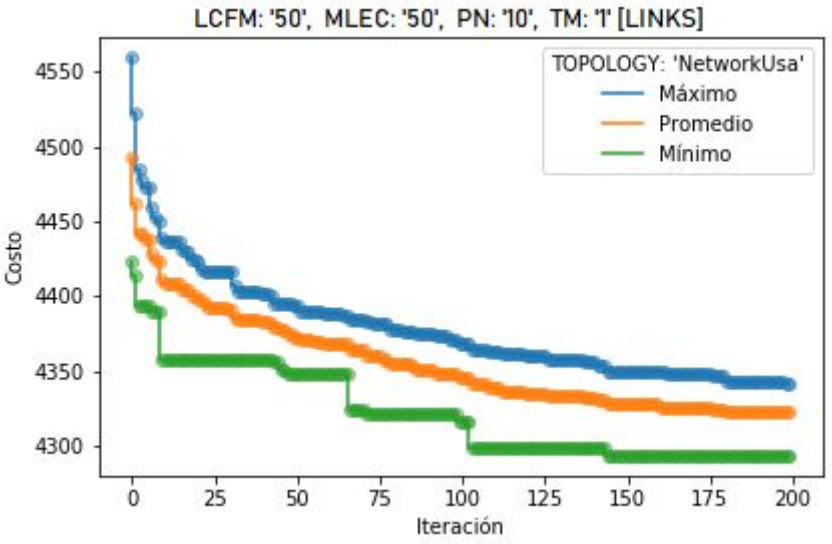

LCFM: '75', MLEC: '50', PN: '5', TM: 'l' [LINKS]
TOPOLOGY: 'CRL'
Máximo
Promedio
Mínimo
Costo
Iteración
2400
2350
2300
2250
2200
0
25
50
75
100
125
150
175
200

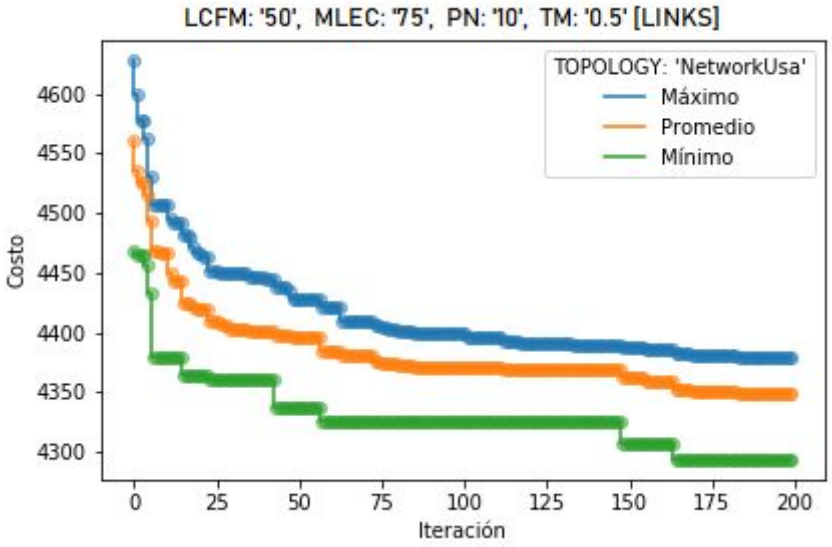


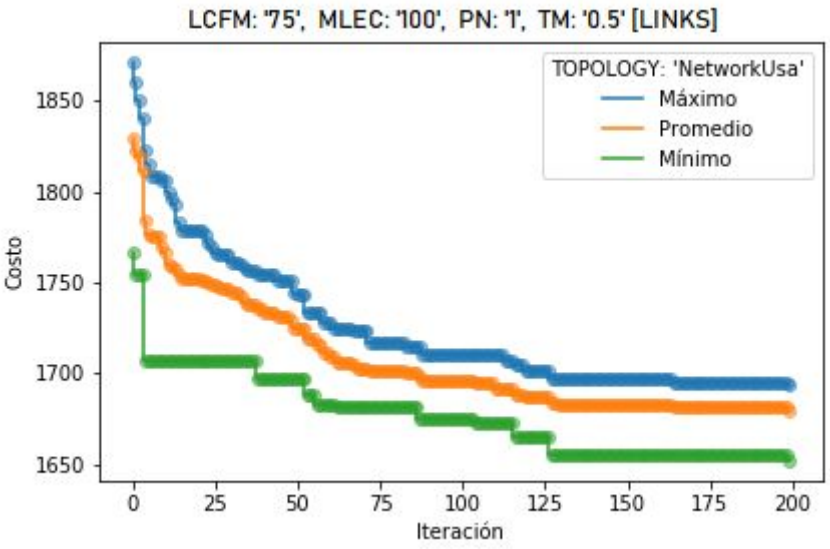
Para la topología Network Usa


LCFM: '100', MLEC: '100', PN: '1', TM: '1' [LINKS]
TOPOLOGY: 'NetworkUsa'
Máximo
Promedio
Mínimo
Costo
Iteración
1850
1825
1800
1775
1750
1725
1700
1675
1650
0
25
50
75
100
125
150
175
200

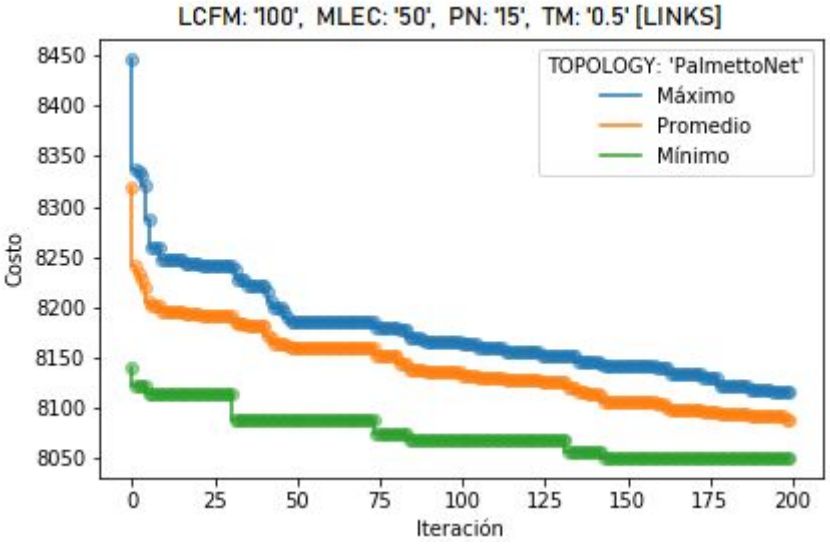

LCFM: '50', MLEC: '50', PN: '10', TM: '0.5' [LINKS]
TOPOLOGY: 'NetworkUsa'
Máximo
Promedio
Mínimo
Costo
Iteración
4600
4550
4500
4450
4400
4350
4300
0
25
50
75
100
125
150
175
200

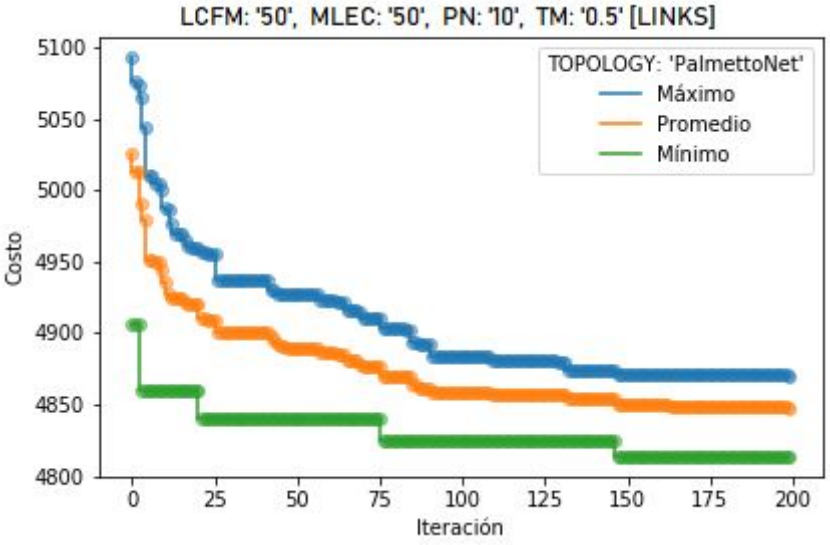

LCFM: '50', MLEC: '50', PN: '10', TM: '1' [LINKS]
TOPOLOGY: 'NetworkUsa'
Máximo
Promedio
Mínimo
Costo
Iteración
4550
4500
4450
4400
4350
4300
0
25
50
75
100
125
150
175
200

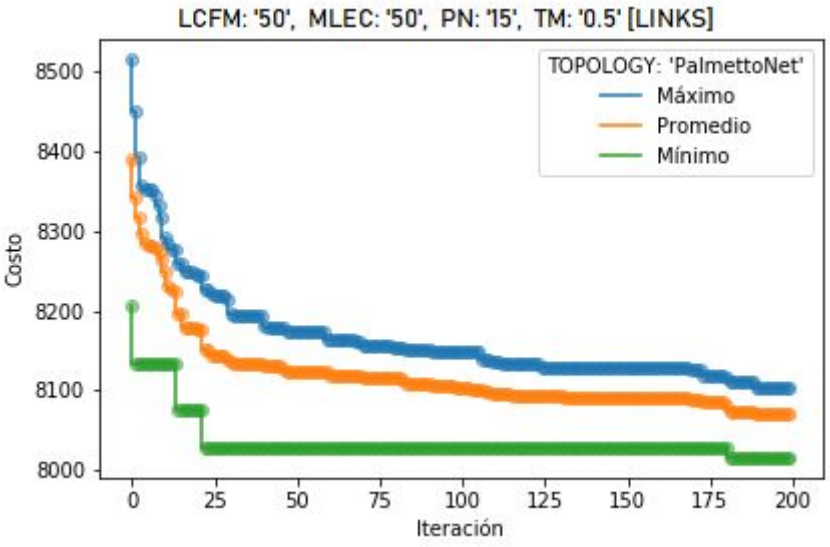

LCFM: '50', MLEC: '75', PN: '10', TM: '0.5' [LINKS]
TOPOLOGY: 'NetworkUsa'
Máximo
Promedio
Mínimo
Costo
Iteración
4600
4550
4500
4450
4400
4350
4300
0
25
50
75
100
125
150
175
200

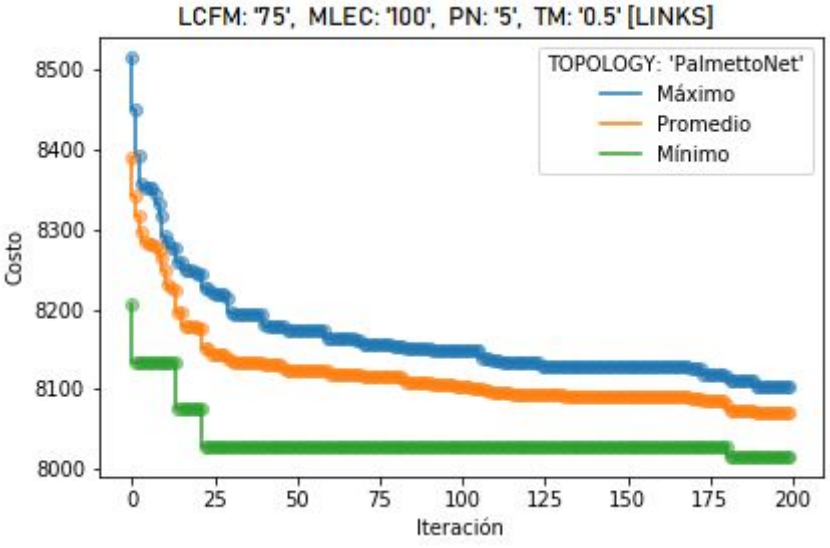


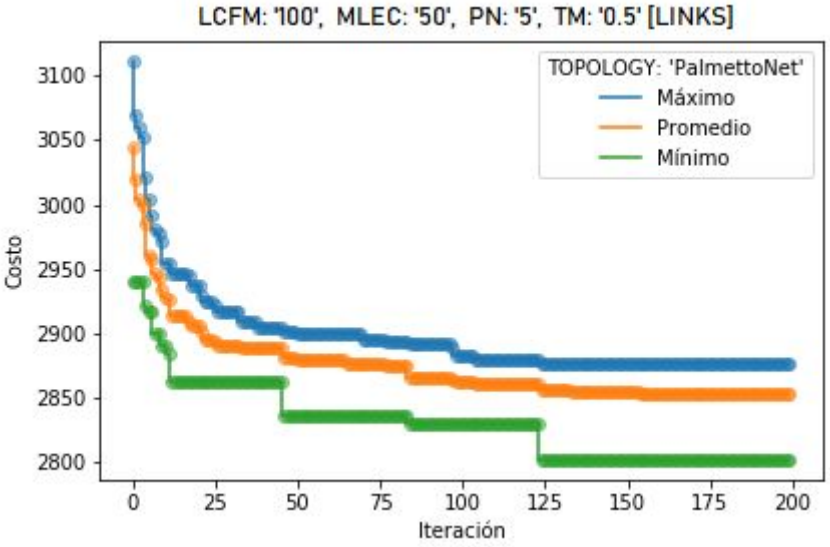
Para la topología PalmetoNet


LCFM: '100', MLEC: '50', PN: '15', TM: '0.5' [LINKS]
TOPOLOGY: 'PalmettoNet'
Máximo
Promedio
Mínimo
Costo
Iteración
8450
8400
8350
8300
8250
8200
8150
8100
8050
0
25
50
75
100
125
150
175
200

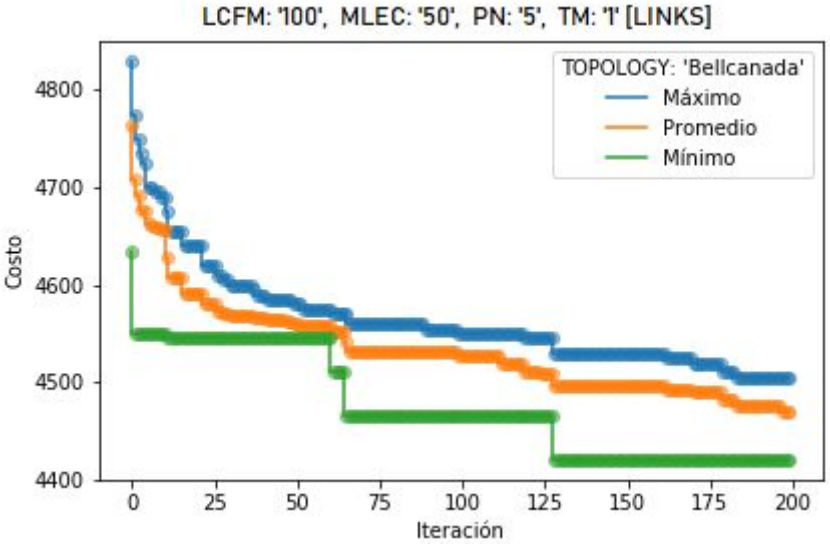

LCFM: '50', MLEC: '50', PN: '10', TM: '0.5' [LINKS]
TOPOLOGY: 'PalmettoNet'
Máximo
Promedio
Mínimo
Costo
Iteración
5100
5050
5000
4950
4900
4850
4800
0
25
50
75
100
125
150
175
200

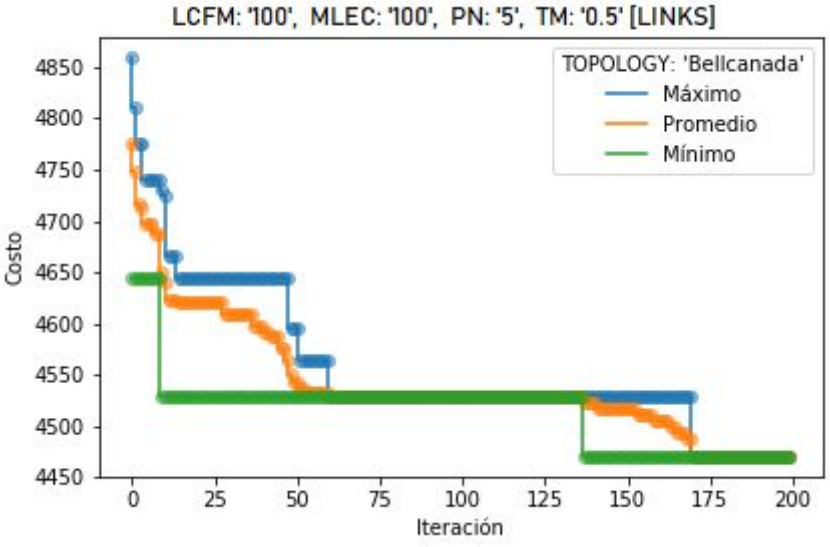

LCFM: '50', MLEC: '50', PN: '15', TM: '0.5' [LINKS]
TOPOLOGY: 'PalmettoNet'
Máximo
Promedio
Mínimo
Costo
Iteración
8500
8400
8300
8200
8100
8000
0
25
50
75
100
125
150
175
200


LCFM: '75', MLEC: '100', PN: '5', TM: '0.5' [LINKS]
TOPOLOGY: 'PalmettoNet'
Máximo
Promedio
Mínimo
Costo
Iteración
8500
8400
8300
8200
8100
8000
0
25
50
75
100
125
150
175
200

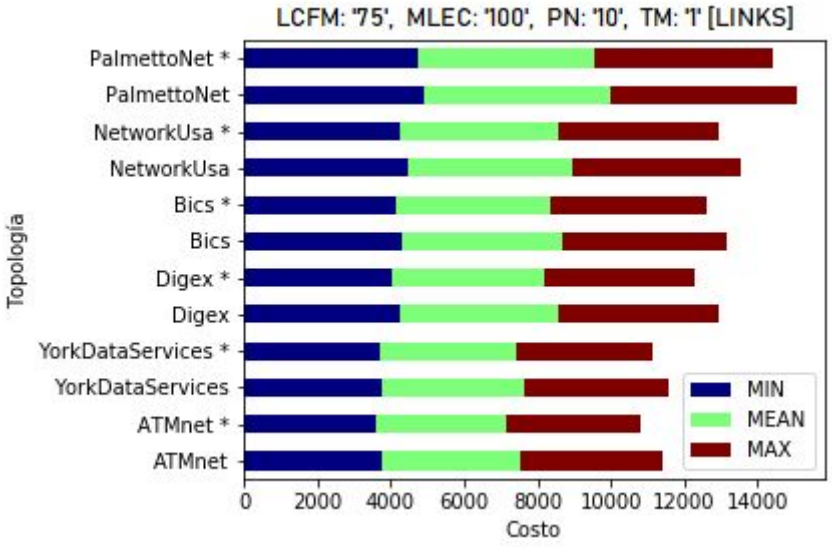


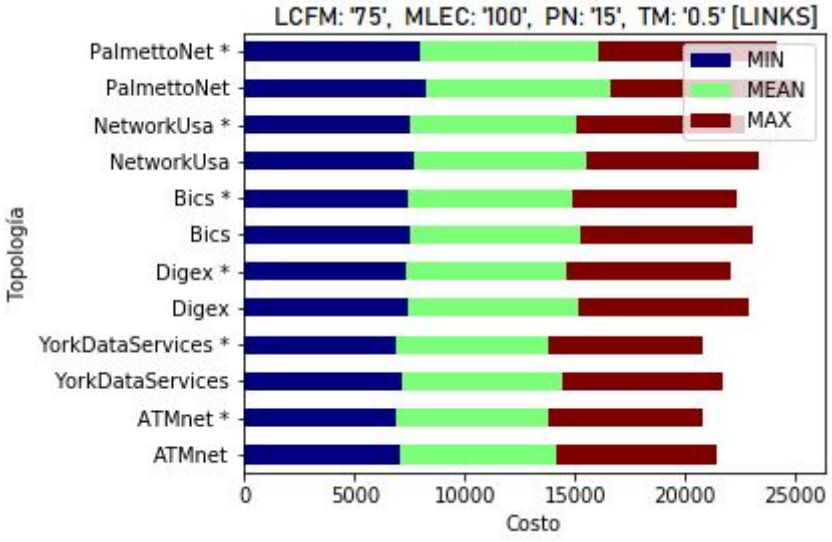
Para la topología Bell Canada

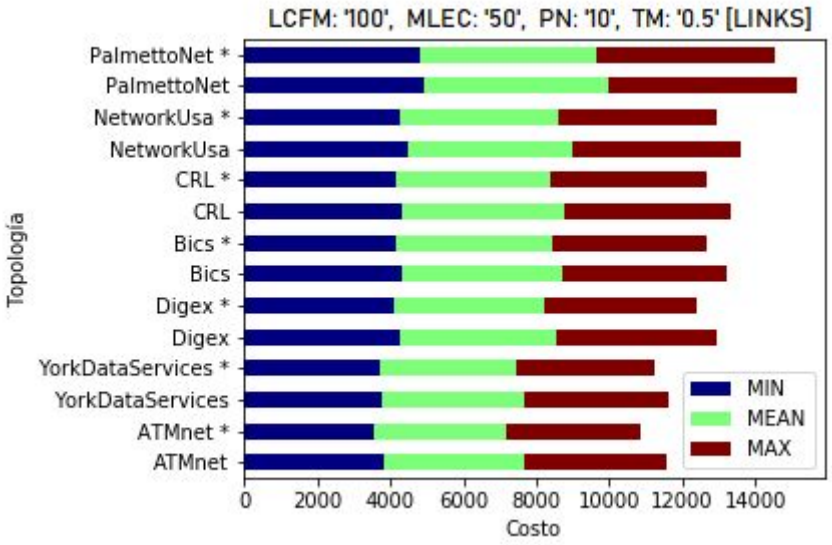

LCFM: '100', MLEC: '50', PN: '5', TM: '1' [LINKS]
TOPOLOGY: 'Bellcanada'
Máximo
Promedio
Mínimo
Costo
Iteración
4800
4700
4600
4500
4400
0
25
50
75
100
125
150
175
200

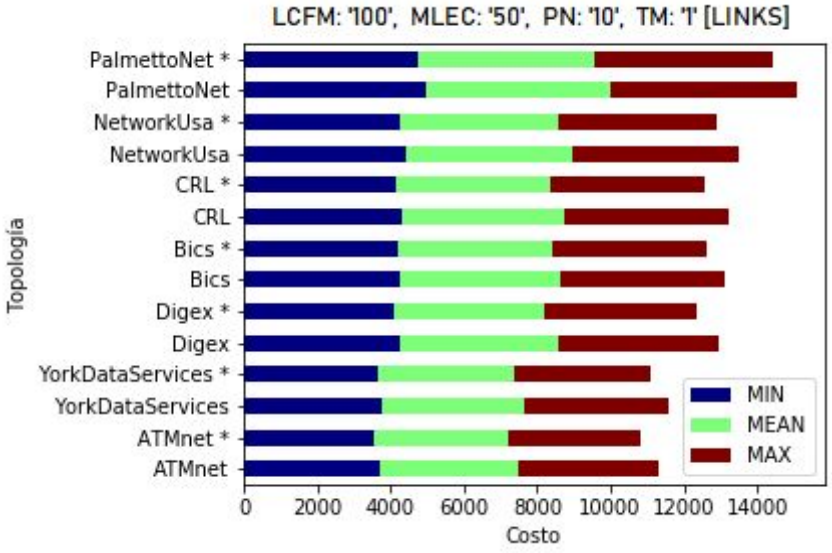

LCFM: '100', MLEC: '100', PN: '5', TM: '0.5' [LINKS]
TOPOLOGY: 'Bellcanada'
Máximo
Promedio
Mínimo
Costo
Iteración
4850
4800
4750
4700
4650
4600
4550
4500
4450
0
25
50
75
100
125
150
175
200

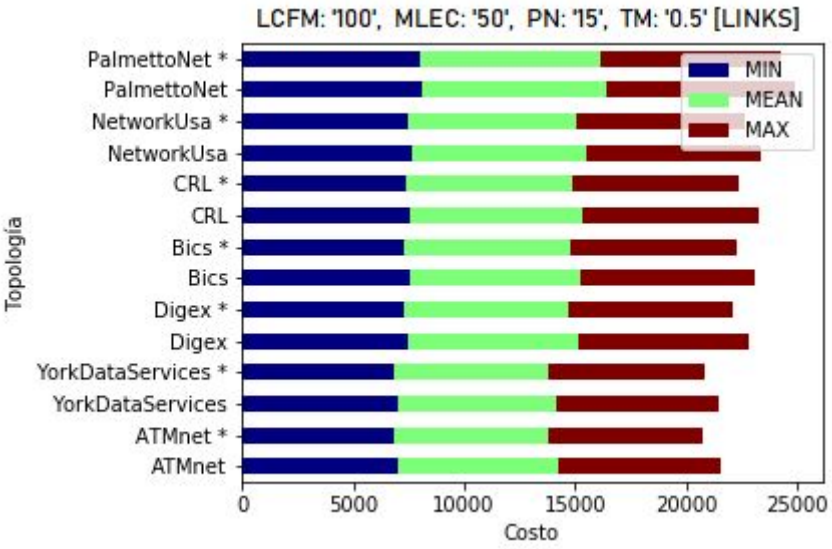
- **Las siguientes gráficas muestran el costo de cada topología antes y después de la optimización luego de 200 generaciones. La optimización se diferencia con el signo * respecto a la original**

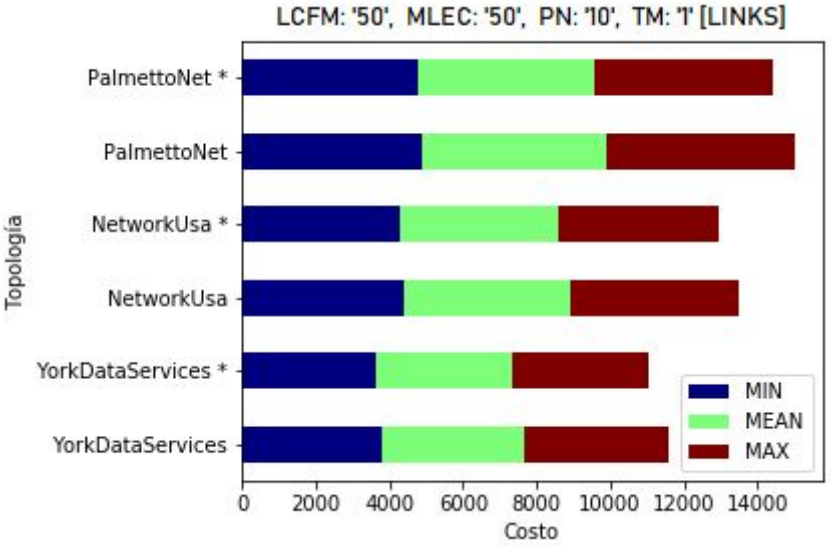


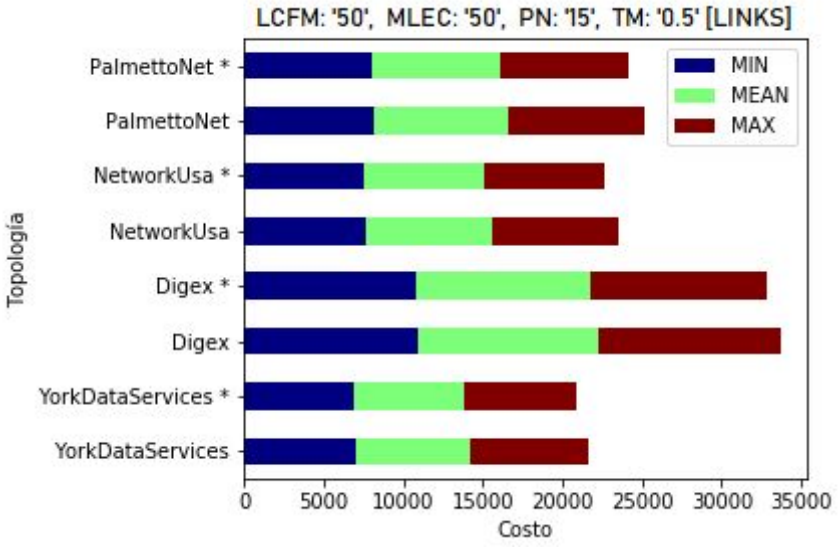


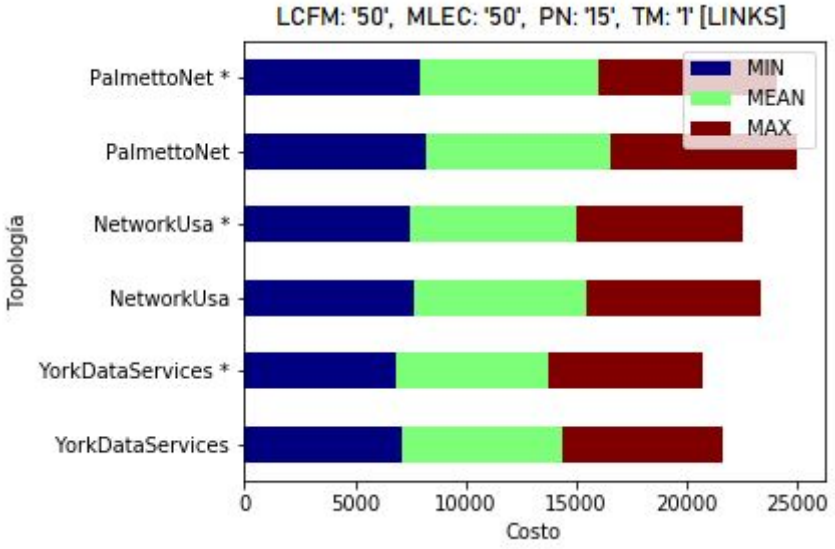

LCFM: '100', MLEC: '50', PN: '10', TM: '0.5' [LINKS]
PalmettoNet *
PalmettoNet
NetworkUsa *
NetworkUsa
CRL *
CRL
Bics *
Bics
Digex *
Digex
YorkDataServices *
YorkDataServices
ATMnet *
ATMnet
Topología
MIN
MEAN
MAX
0
2000
4000
6000
8000
10000
12000
14000
Costo

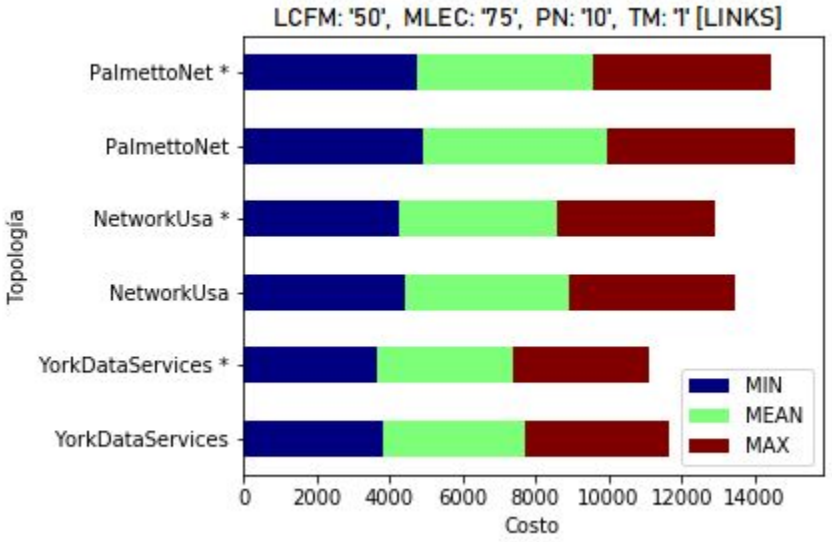

LCFM: '100', MLEC: '50', PN: '10', TM: 'I' [LINKS]
PalmettoNet *
PalmettoNet
NetworkUsa *
NetworkUsa
CRL *
CRL
Bics *
Bics
Digex *
Digex
YorkDataServices *
YorkDataServices
ATMnet *
ATMnet
Topología
MIN
MEAN
MAX
0
2000
4000
6000
8000
10000
12000
14000
Costo

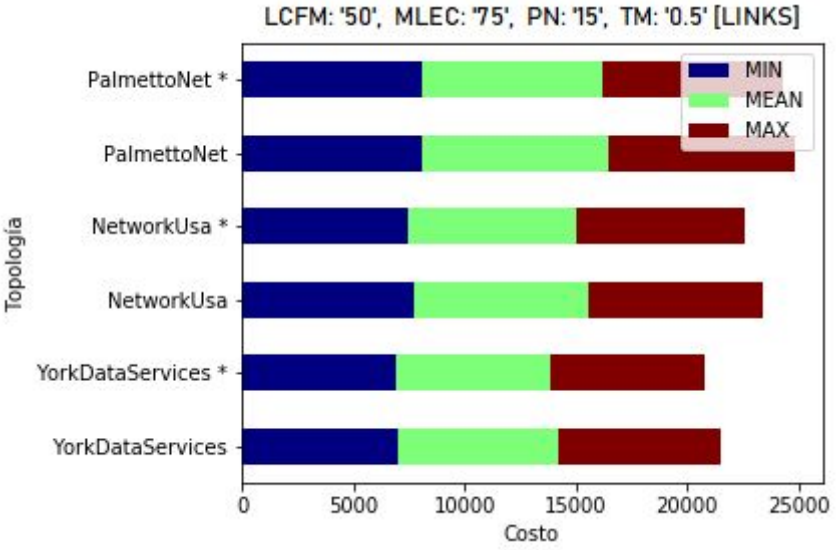

LCFM: '100', MLEC: '50', PN: '15', TM: '0.5' [LINKS]
PalmettoNet *
PalmettoNet
NetworkUsa *
NetworkUsa
CRL *
CRL
Bics *
Bics
Digex *
Digex
YorkDataServices *
YorkDataServices
ATMnet *
ATMnet
MIN
MEAN
MAX
Topología
0
5000
10000
15000
20000
25000
Costo

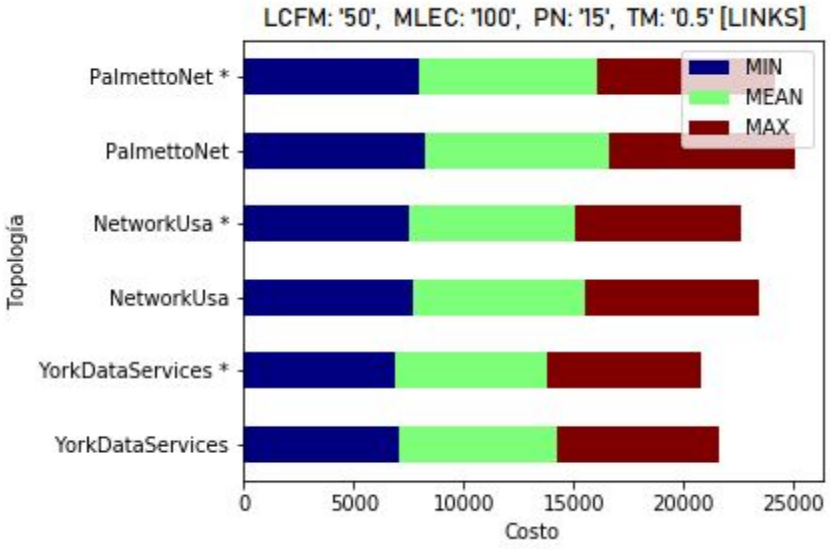

LCFM: '50', MLEC: '50', PN: '10', TM: '1' [LINKS]
PalmettoNet *
PalmettoNet
NetworkUsa *
NetworkUsa
YorkDataServices *
YorkDataServices
MIN
MEAN
MAX
Topología
0
2000
4000
6000
8000
10000
12000
14000
Costo

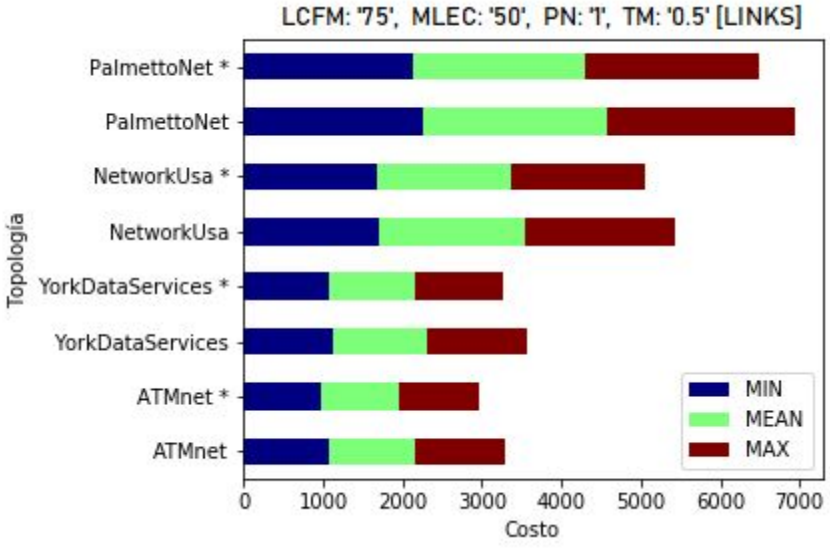

LCFM: '50', MLEC: '50', PN: '15', TM: '0.5' [LINKS]
MIN
MEAN
MAX
PalmettoNet *
PalmettoNet
NetworkUsa *
NetworkUsa
Digex *
Digex
YorkDataServices *
YorkDataServices
Topología
0
5000
10000
15000
20000
25000
30000
35000
Costo


LCFM: '50', MLEC: '50', PN: '15', TM: '1' [LINKS]
MIN
MEAN
MAX
PalmettoNet *
PalmettoNet
NetworkUsa *
NetworkUsa
YorkDataServices *
YorkDataServices
Topología
0
5000
10000
15000
20000
25000
Costo

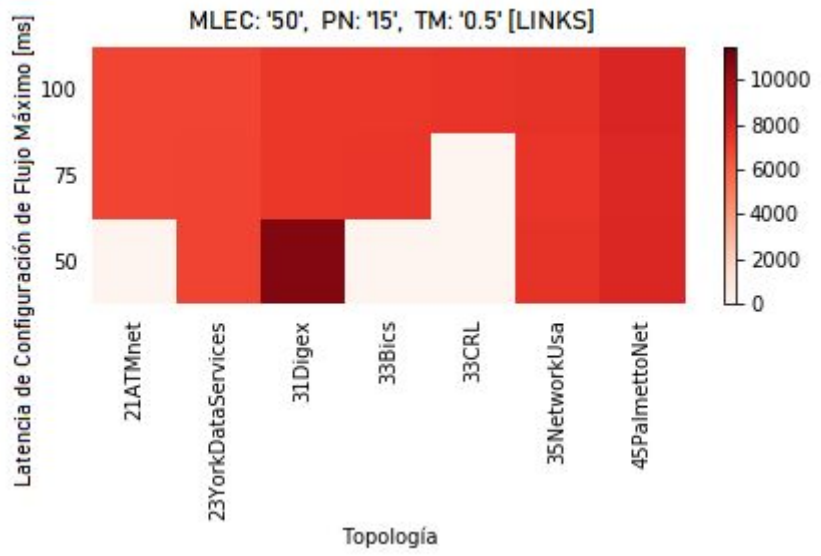

LCFM: '50', MLEC: '75', PN: '10', TM: '1' [LINKS]
PalmettoNet *
PalmettoNet
NetworkUsa *
NetworkUsa
YorkDataServices *
YorkDataServices
Topología
MIN
MEAN
MAX
0
2000
4000
6000
8000
10000
12000
14000
Costo

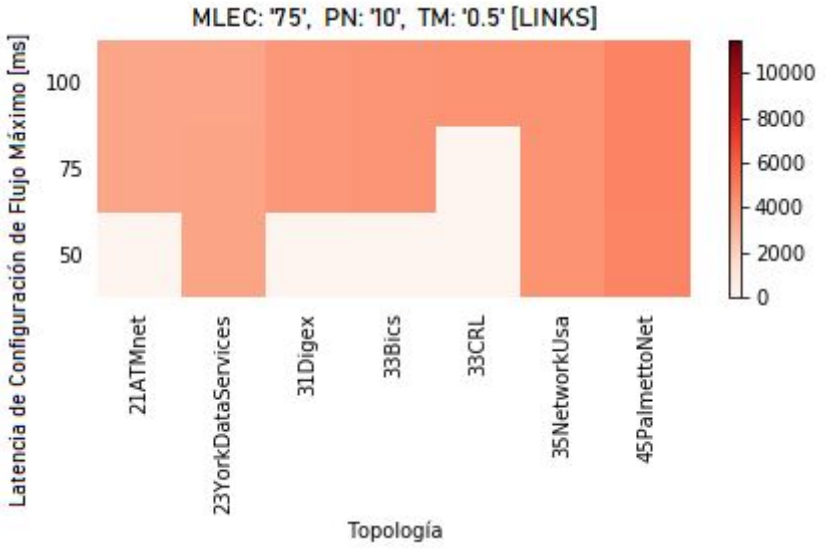

LCFM: '50', MLEC: '75', PN: '15', TM: '0.5' [LINKS]
PalmettoNet *
PalmettoNet
NetworkUsa *
NetworkUsa
YorkDataServices *
YorkDataServices
Topología
MIN
MEAN
MAX
0
5000
10000
15000
20000
25000
Costo

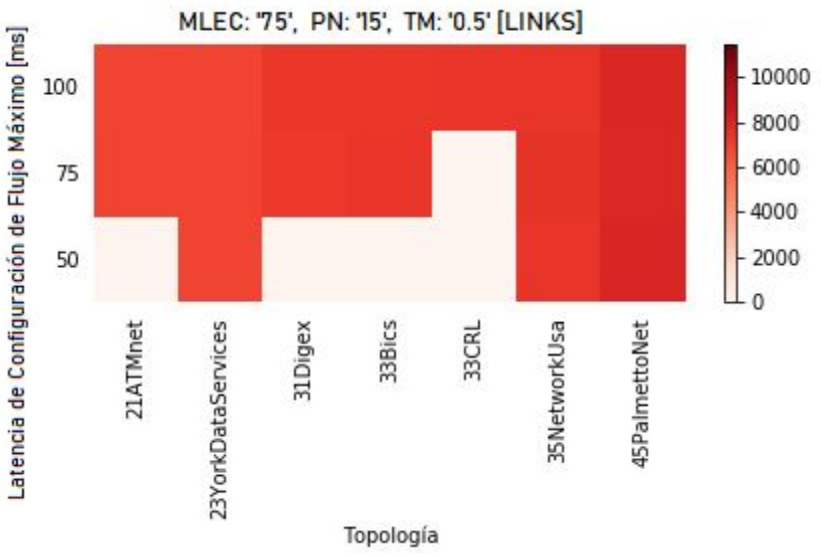

LCFM: '50', MLEC: '100', PN: '15', TM: '0.5' [LINKS]
PalmettoNet *
PalmettoNet
NetworkUsa *
NetworkUsa
YorkDataServices *
YorkDataServices
Topología
MIN
MEAN
MAX
0
5000
10000
15000
20000
25000
Costo

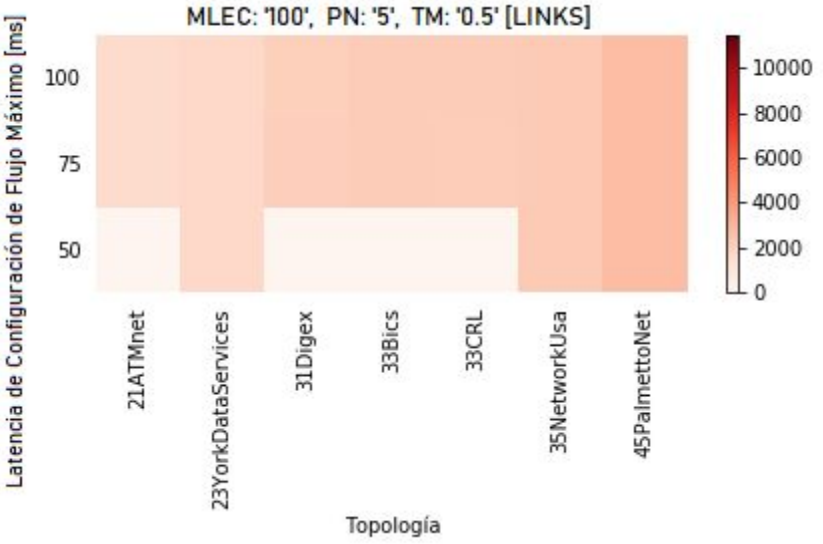

LCFM: '75', MLEC: '50', PN: '1', TM: '0.5' [LINKS]
PalmettoNet *
PalmettoNet
NetworkUsa *
NetworkUsa
YorkDataServices *
YorkDataServices
ATMnet *
ATMnet
Topología
MIN
MEAN
MAX
0
1000
2000
3000
4000
5000
6000
7000
Costo

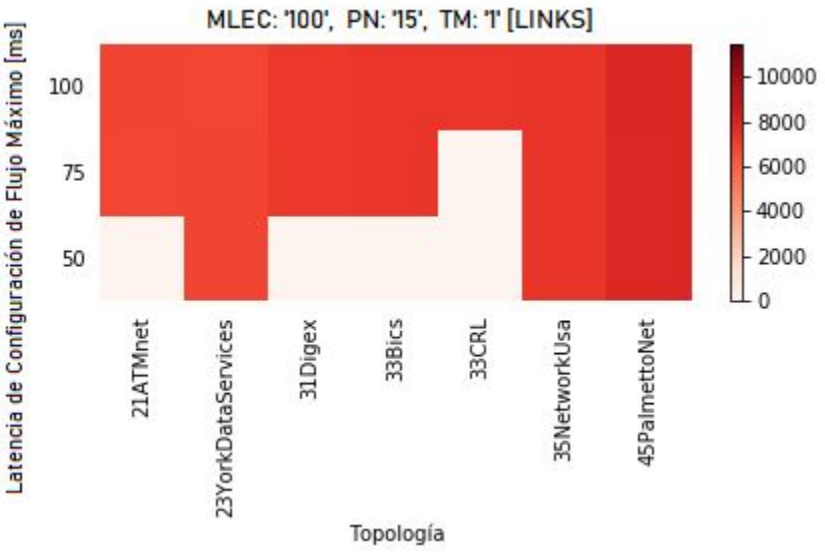
- **Las siguientes gráficas muestran el costo de cada topología vs el LCFM**

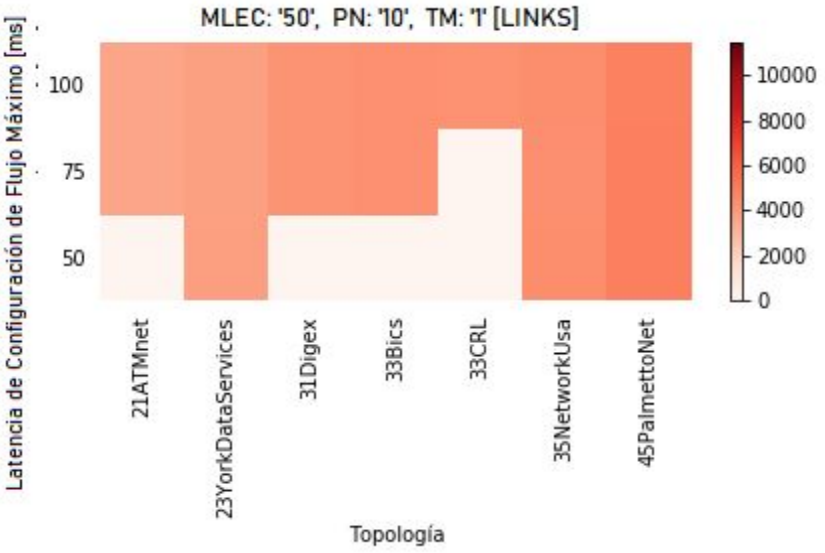


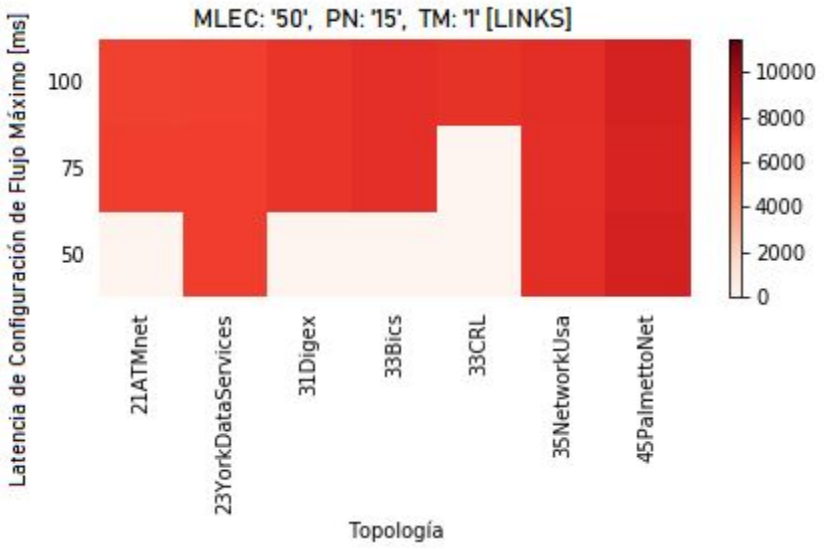


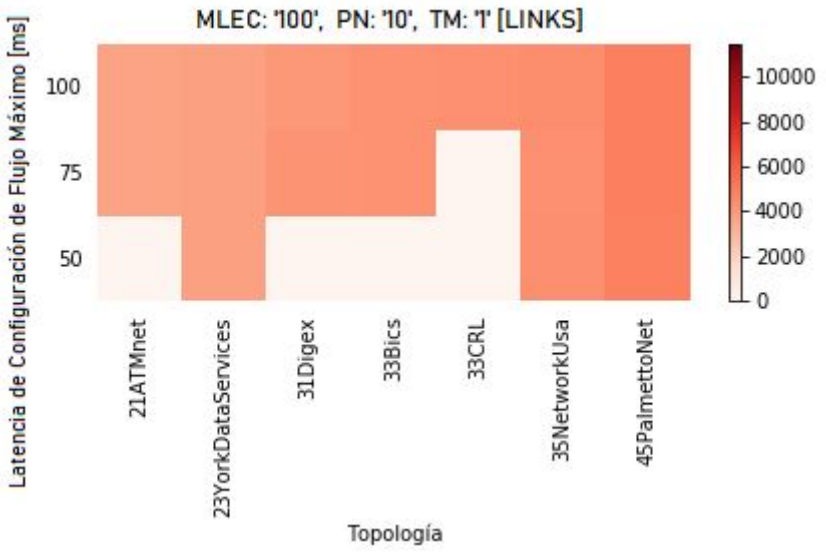

MLEC: '75', PN: '15', TM: '0.5' [LINKS]
Latencia de Configuración de Flujo Máximo [ms]
100
75
50
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología
10000
8000
6000
4000
2000
0

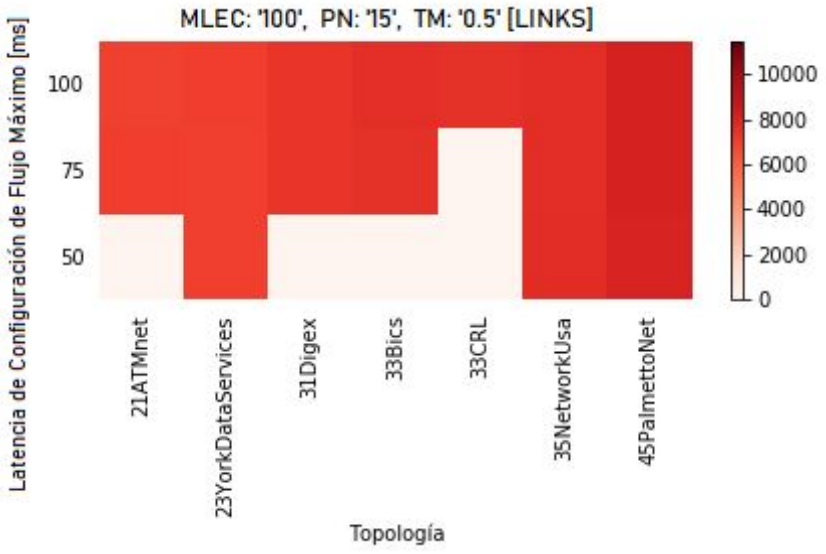

MLEC: '100', PN: '5', TM: '0.5' [LINKS]
Latencia de Configuración de Flujo Máximo [ms]
100
75
50
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología
10000
8000
6000
4000
2000
0

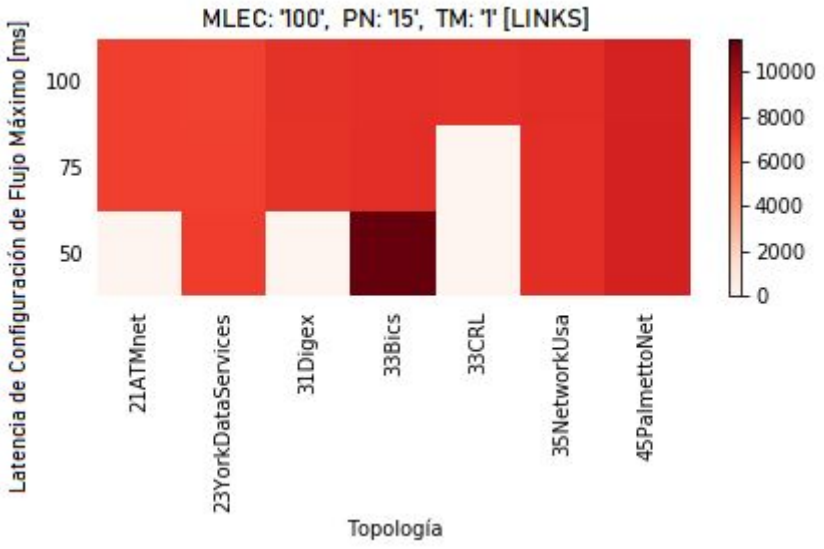

MLEC: '100', PN: '15', TM: 'l' [LINKS]
Latencia de Configuración de Flujo Máximo [ms]
100
75
50
10000
8000
6000
4000
2000
0
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología


MLEC: '50', PN: '10', TM: 'l' [LINKS]
Latencia de Configuración de Flujo Máximo [ms]
100
75
50
10000
8000
6000
4000
2000
0
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología

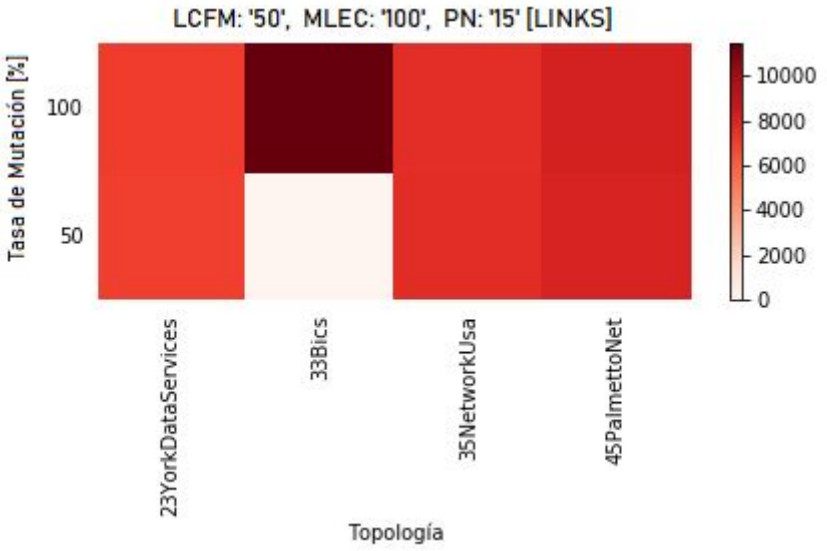

MLEC: '50', PN: '15', TM: '1' [LINKS]
Latencia de Configuración de Flujo Máximo [ms]
100
75
50
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología
10000
8000
6000
4000
2000
0

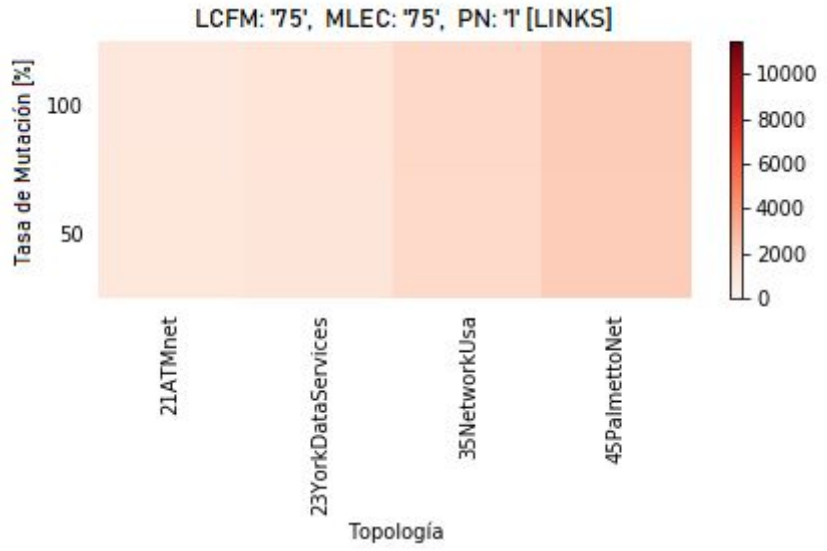

MLEC: '100', PN: '10', TM: '1' [LINKS]
Latencia de Configuración de Flujo Máximo [ms]
100
75
50
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología
10000
8000
6000
4000
2000
0

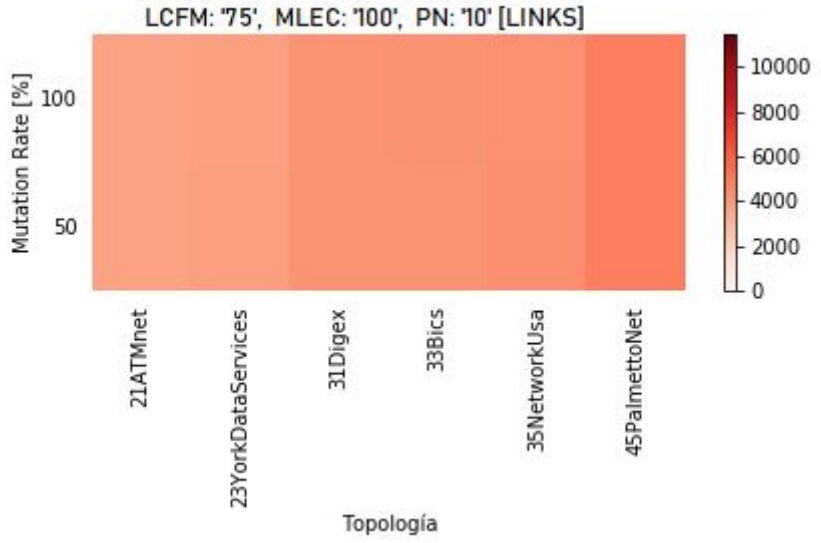

MLEC: '100', PN: '15', TM: '0.5' [LINKS]
Latencia de Configuración de Flujo Máximo [ms]
100
75
50
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología
10000
8000
6000
4000
2000
0

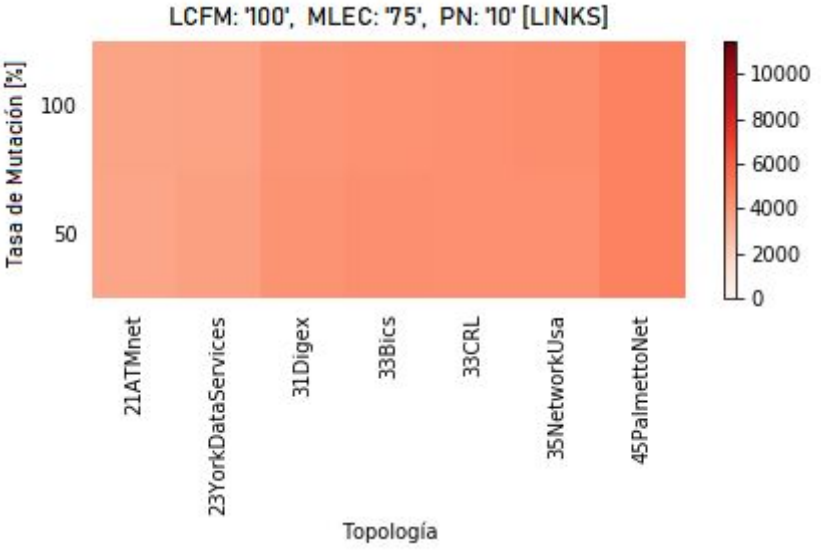

MLEC: '100', PN: '15', TM: '1' [LINKS]
Latencia de Configuración de Flujo Máximo [ms]
100
75
50
10000
8000
6000
4000
2000
0
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología

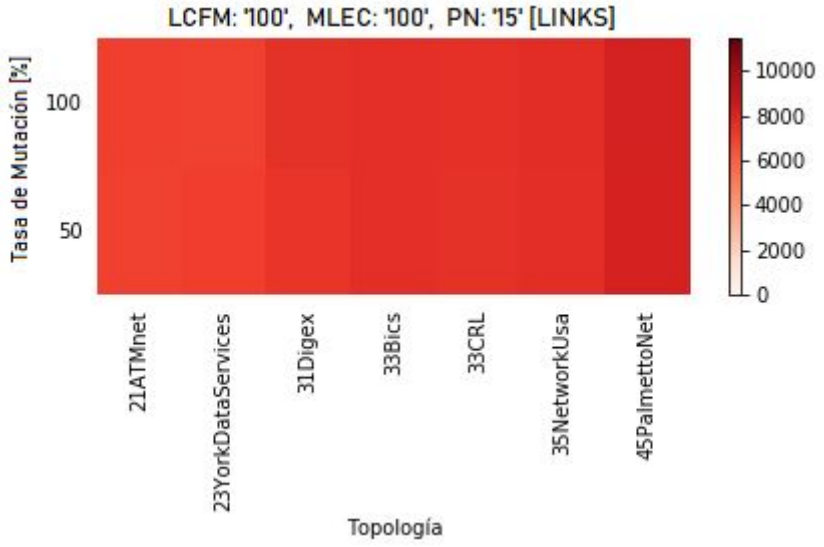
- **Las siguientes gráficas muestran el costo de cada topología vs el TM(0.5 y 1)**

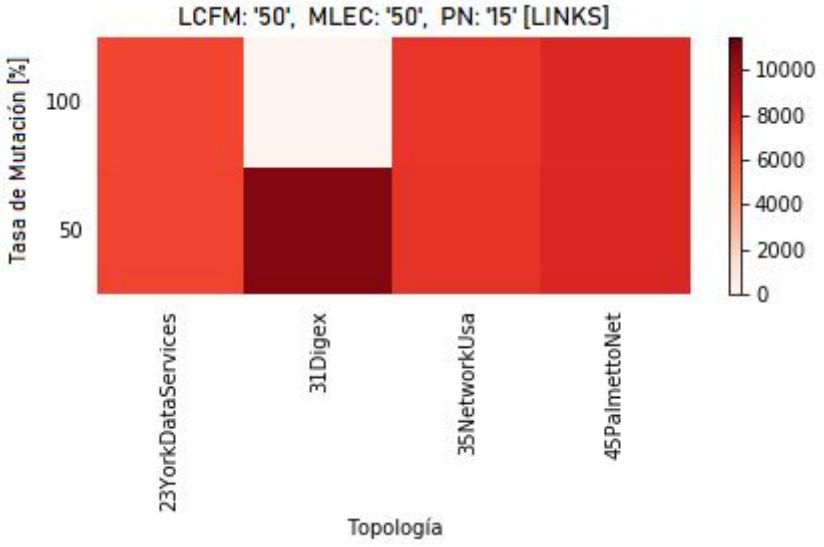


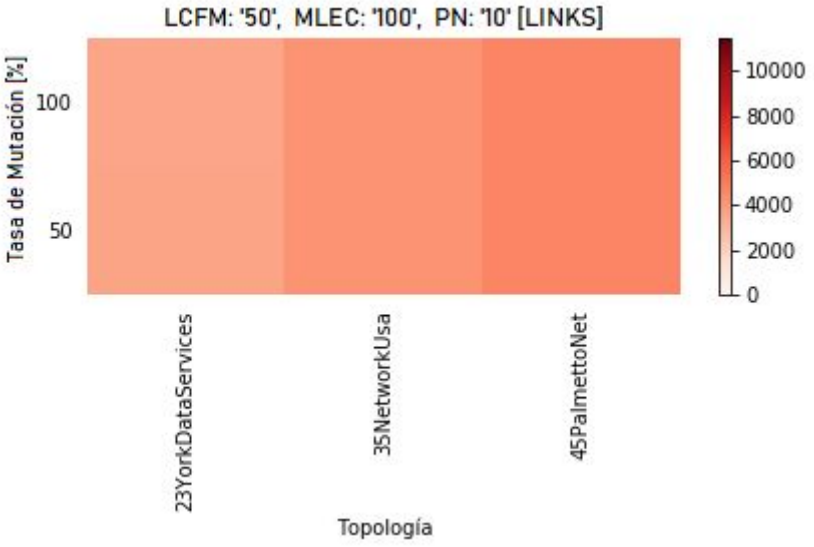


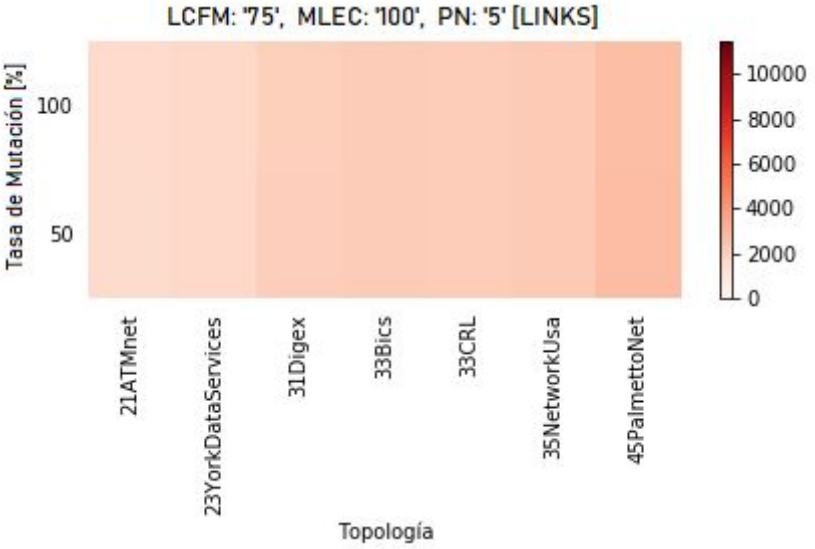

LCFM: '75', MLEC: '100', PN: '10' [LINKS]
Mutation Rate [%]
100
50
10000
8000
6000
4000
2000
0
21ATMnet
23YorkDataServices
31Digex
33Bics
35NetworkUsa
45PalmettoNet
Topología

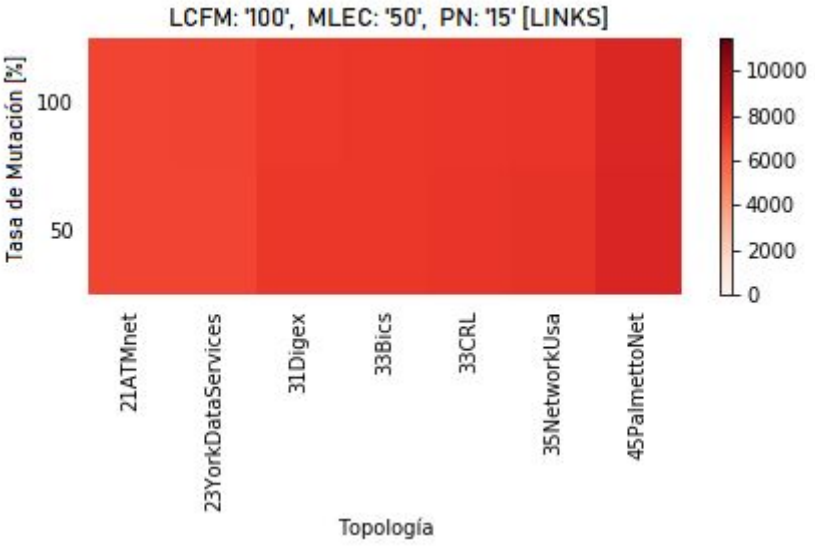

LCFM: '100', MLEC: '75', PN: '10' [LINKS]
Tasa de Mutación [%]
100
50
10000
8000
6000
4000
2000
0
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología

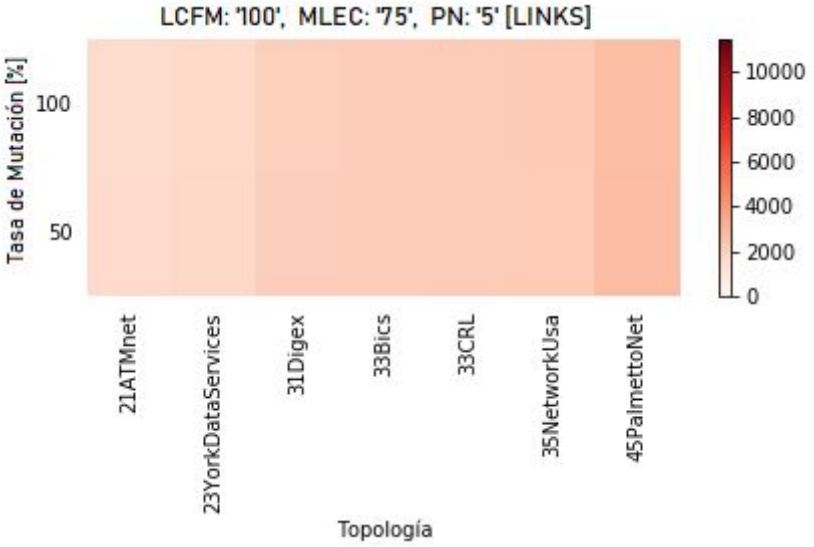

LCFM: '100', MLEC: '100', PN: '15' [LINKS]
Tasa de Mutación [%]
100
50
10000
8000
6000
4000
2000
0
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología

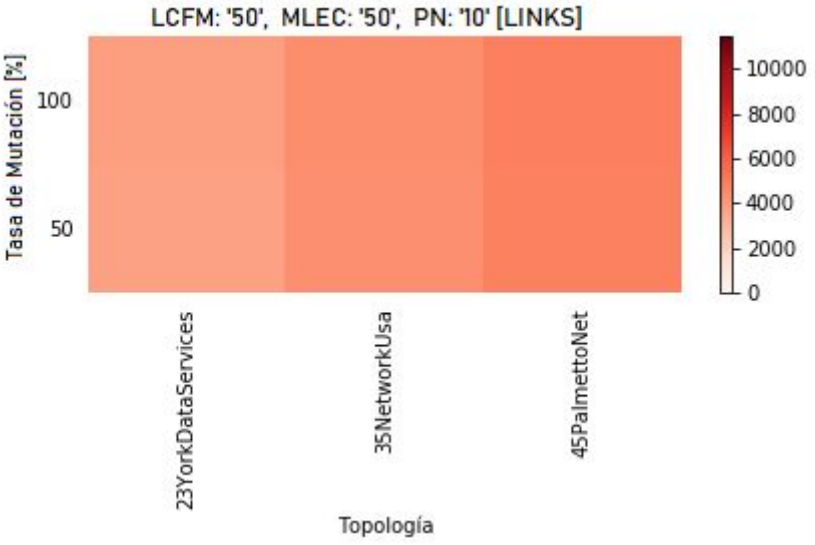

LCFM: '50', MLEC: '50', PN: '15' [LINKS]
Tasa de Mutación [%]
100
50
10000
8000
6000
4000
2000
0
23YorkDataServices
31Digex
35NetworkUsa
45PalmettoNet
Topología

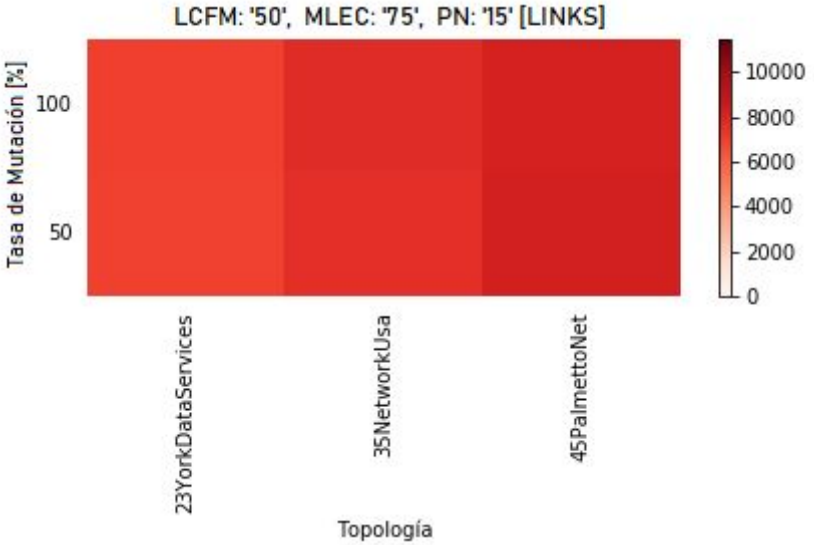

LCFM: '50', MLEC: '100', PN: '10' [LINKS]
Tasa de Mutación [%]
100
50
10000
8000
6000
4000
2000
0
23YorkDataServices
35NetworkUsa
45PalmettoNet
Topología


LCFM: '75', MLEC: '100', PN: '5' [LINKS]
Tasa de Mutación [%]
100
50
10000
8000
6000
4000
2000
0
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología


LCFM: '100', MLEC: '50', PN: '15' [LINKS]
Tasa de Mutación [%]
100
50
10000
8000
6000
4000
2000
0
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología


LCFM: '100', MLEC: '75', PN: '5' [LINKS]
Tasa de Mutación [%]
100
50
10000
8000
6000
4000
2000
0
21ATMnet
23YorkDataServices
31Digex
33Bics
33CRL
35NetworkUsa
45PalmettoNet
Topología


LCFM: '50', MLEC: '50', PN: '10' [LINKS]
Tasa de Mutación [%]
100
50
10000
8000
6000
4000
2000
0
23YorkDataServices
35NetworkUsa
45PalmettoNet
Topología


LCFM: '50', MLEC: '75', PN: '15' [LINKS]
Tasa de Mutación [%]
100
50
10000
8000
6000
4000
2000
0
23YorkDataServices
35NetworkUsa
45PalmettoNet
Topología

Referencias

[1] B.A.A. Nunes, M. Mendonca, X.N. Nguyen, K. Obraczka and T. Turletti, '"A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," IEEE Communications Surveys & Tutorials, vol. 16, no. 3, pp. 1617-1634, 2014.

[2] M.F. Bari, A.R. Roy, S.R. Chowdhury, Q. Zhang, M.F. Zhani, R. Ahmed and R. Boutaba, '"Dynamic Controller Provisioning in Software Defined Networks," Proceedings of the 9th International Conference on Network and Service Management, pp. 18-25, 2013.

[3] F. Wang, H. Wang, B. Lei and W. Ma, '"A Research on High-Performance SDN Controller," International Conference on Cloud Computing and Big Data, pp. 168-174, 2014.

[4] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner and P. Tran-Gia, '"Pareto-optimal resilient controller placement in SDN-based core networks," Proceedings of the 2013 25th International Teletraffic Congress (ITC), pp. 1-9.

[5] Y. Fan, Y. Xia, W. Liang and X. Zhang, '"Latency-Aware Reliable Controller Placements in SDNs,", pp. 152-162, 2013.

[6] B. Heller, R. Sherwood, and N. McKeown, '"The controller placement problem,", August 13, pp. 7-12, 2012.

[7] E.K. Ali, M. Manel and Y. Habib, '"An Efficient MPLS-Based Source Routing Scheme in Software-Defined Wide Area Networks (SD-WAN)," IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), pp. 1205-1211, 2015.

[8] Brian Underdahl and Gary Kinghorn, '"Software Defined Networking for Dummies, Cisco Special Edition,", 2015.

[9] L. Mamushiane, A. Lysko and S. Dlamini, '"A comparative evaluation of the performance of popular SDN controllers," Wireless Days (WD), pp. 54-59, 2018.

[10] Aitana Vidal Esmorís, '"Algoritmos heurísticos en optimización,", pp. 50-54, 2013.

[11] Clinton Sheppard, '"Algoritmos genéticos con Python,", pp. 115-123, 2017.

[12] E.R. Jimson, K. Nisar and bin Ahmad Hijazi, M H, '"Bandwidth management using software defined network and comparison of the throughput performance with traditional network," International Conference on Computer and Drone Applications (IConDA), pp. 71-76, 2017.

[13] Y. Cui, Y. Chen, J. Liu, Y.l. Lee, J. Wu and X. Wang, '"State management in IPv4 to IPv6 transition," IEEE Network, vol. 29, no. 6, pp. 48-53, 2015.

[14] Raghavan, Barath and Casado, Mart and Koponen, Teemu and Ratnasamy, Sylvia and Ghodsi, Ali and Shenker, Scott, '"Software-defined Internet Architecture: Decoupling Architecture from Infrastructure," Proceedings of the 11th ACM Workshop on Hot Topics in Networks, pp. 43-48, 2012.

[15] T. Koponen, '"Software is the future of networking," ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 135, 2012.

[16] H. Kim and N. Feamster, '"Improving network management with software defined networking," IEEE Communications Magazine, vol. 51, no. 2, pp. 114-119, 2013.

[17] L. Tancevski, '"SDN concept: From theory to network implementation," OFC, pp. 1-3, 2014.

[18] E. Tantik and R. Anderl, '"Concept of the asset administration shell as a software-defined system," Fifth International Conference on Software Defined Systems (SDS), pp. 52-58, 2018.

[19] C. Banse and S. Rangarajan, '"A Secure Northbound Interface for SDN Applications," IEEE Trustcom/BigDataSE/ISPA, vol. 1, pp. 834-839, 2015.

[20] X. Lu and Y. Xu, '"SFabric: A scalable SDN based large layer 2 data center network fabric," IEEE 23rd International Symposium on Quality of Service (IWQoS), pp. 57-58, 2015.

[21] McKeown, Nick and Anderson, Tom and Balakrishnan, '"OpenFlow: Enabling Innovation in Campus Networks," SIGCOMM Comput. Commun. Rev., April, pp. 69-74, 2008.

[22] S. Raghul, T. Subashri and K.R. Vimal, '"Literature survey on traffic-based server load balancing using SDN and open flow," Fourth International Conference on Signal Processing, Communication and Networking (ICSCN), pp. 1-6, 2017.

[23] C. Wang and S. Yan, '"Scaling SDN network with self-adjusting architecture," IEEE International Conference on Electronic Information and Communication Technology (ICEICT), pp. 116-120, 2016.

[24] Z. Bozakov and A. Rizk, '"Taming SDN Controllers in Heterogeneous Hardware Environments," Second European Workshop on Software Defined Networks, pp. 50-55, 2013.

[25] F. Alencar, M. Santos, M. Santana and S. Fernandes, '"How Software Aging affects SDN: A view on the controllers," Global Information Infrastructure and Networking Symposium (GIIS), pp. 1-6, 2014.

[26] R. Khondoker, A. Zaalouk, R. Marx and K. Bayarou, '"Feature-based comparison and selection of Software Defined Networking (SDN) controllers," World Congress on Computer Applications and Information Systems (WCCAIS), pp. 1-7, 2014.

[27] G. Yao, J. Bi, Y. Li and L. Guo, '"On the Capacitated Controller Placement Problem in Software Defined Networks," IEEE Communications Letters, vol. 18, no. 8, pp. 1339-1342, 2014.

[28] S. Guo, S. Yang, Q. Li and Y. Jiang, '"Towards Controller Placement for robust Software-Defined Networks," IEEE 34th International Performance Computing and Communications Conference (IPCCC), pp. 1-8, 2015.

[29] D. Hock, S. Gebert, M. Hartmann, T. Zinner and P. Tran-Gia, '"POCO-framework for Pareto-optimal resilient controller placement in SDN-based core networks," IEEE Network Operations and Management Symposium (NOMS), pp. 1-2, 2014.

[30] A. I. Irawan, M. Rahayu, F. Nisa and N. R. Syambas, '"Network migration to SDN using Pareto Optimal Resilience Controller (POCO): Case study in the UPI network," 9th International Conference on Telecommunication Systems Services and Applications (TSSA), pp. 1-6, 2015.

[31] S. Kirkpatrick, C.D. Gelatt Jr and M.P. Vecchi, '"Optimization by Simulated Annealing," Science, vol. 220, no. 4598, May 13, pp. 671-680, 2010.

[32] D. Hock, M. Hartmann, S. Gebert, T. Zinner and P. Tran-Gia, '"POCO-PLC: Enabling dynamic pareto-optimal resilient controller placement in SDN networks," IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 115-116, 2014.

[33] Cormen Thomas, Leiserson Charles and Rivest Ronald, '"Introduction to Algorithms,"pp. 25-70, 2009.

[34] P.J. van Laarhoven, E H Aarts, '"Simulated Annealing: Theory and Applications,"pp. 1-9, 2013.

[35] J. Brownlee, '"Clever Algorithms: Nature-inspired Programming Recipes,", 2011.

[36] M. Obadia, M. Bouet, J. Rougier and L. Iannone, '"A greedy approach for minimizing SDN control overhead," Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft), pp. 1-5, 2015.

[37] Y. HU, W. WANG, X. GONG, X. QUE and S. CHENG, '"On the placement of controllers in software-defined networks," The Journal of China Universities of Posts and Telecommunications, vol. 19, no. Supplement 2, pp. 92-171. 2012.
[38] J. Liu, J. Liu and R. Xie, '"Reliability-based controller placement algorithm in software defined networking," Computer Science and Information Systems, vol. 13, no. 2, pp. 547-560, 2016.
[39] ul Huque, M T I, W. Si, G. Jourjon and V. Gramoli, '"Large-Scale Dynamic Controller Placement," IEEE Transactions on Network and Service Management, vol. 14, no. 1, pp. 63-76, 2017.
[40] L. Liao and V.C.M. Leung, '"Genetic algorithms with particle swarm optimization based mutation for distributed controller placement in SDNs," IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1-6, 2017.
[41] B. Webb, '"Swarm Intelligence: From Natural to Artificial Systems,", vol. 14, 2002.
[42] G. Wang, Y. Zhao, J. Huang and W. Wang, '"The Controller Placement Problem in Software Defined Networking: A Survey," IEEE Network, vol. 31, no. 5, pp. 21-27, 2017.
[43] K. He, J. Khalid, S. Das, A. Gember-Jacobson, C. Prakash, A. Akella, L.E. Li and M. Thottan, '"The Controller Placement Problem in Software Defined Networks," ACM SIGMETRICS Performance Evaluation Review, vol. 43, no. 1, Jun 15, pp. 435-436, 2017.
[44] S. Lange, S. Gebert, J. Spoerhase, P. Rygielski, T. Zinner, S. Kounev and P. Tran-Gia, '"Specialized Heuristics for the Controller Placement Problem in Large Scale SDN Networks," 27th International Teletraffic Congress, pp. 210-218, 2015.
[45] L. Yao, P. Hong, W. Zhang, J. Li and D. Ni, '"Controller placement and flow based dynamic management problem towards SDN," IEEE International Conference on Communication Workshop (ICCW), pp. 363-368, 2015.
[46] H.K. Rath, V. Revoori, S.M. Nadaf and A. Simha, '"Optimal controller placement in Software Defined Networks (SDN) using a non-zero-sum game," Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, pp. 1-6, 2014.
[47] L.F. Müller, R.R. Oliveira, M.C. Luizelli, L.P. Gaspary and M.P. Barcellos, '"Survivor: An enhanced controller placement strategy for improving SDN survivability," IEEE Global Communications Conference, pp. 1909-1915, 2014.

[48] S. Lin and B.W. Kernighan, '"An Effective Heuristic Algorithm for the Traveling-Salesman Problem," Oper.Res., vol. 21, no. 2, pp. 498-516, 1973.
[49] A. Charnes and W.W. Cooper, '"Programming with linear fractional functionals," Naval Research Logistics, vol. 9, no. 3, pp. 181-186, 1962.
[50] Daniel Bienstock and George Nemhauser, '"Integer Programming and Combinatorial Optimization,", pp. 443, 2004.
[51] Pandian Vasant, '"Meta-Heuristic Optimization Algorithms in Engineering, Business, Economics, and Finance,", 2013.
[52] H. Samarghandi and K. Eshghi, '"An efficient tabu algorithm for the single row facility layout problem," Eur.J.Oper.Res., vol. 205, no. 1, pp. 98-105, 2010.
[53] D. Fallis, '"The Reliability of Randomized Algorithms," The British Journal for the Philosophy of Science, vol. 51, no. 2, pp. 255-271, 2000.
[54] M. Niazi and A. Hussain, '"Agent-based computing from multi-agent systems to agent-based models: a visual survey," Scientometrics, vol. 89, no. 2, pp. 479, 2017.
[55] Dorigo Marco and Stutzle Thomas, '"Ant Colony Optimization,", 2004.
[56] X. Yang and J. Wang, '"Application of improved ant colony optimization algorithm on traveling salesman problem," Chinese Control and Decision Conference (CCDC), pp. 2156-2160, 2016.
[57] B. Chen and G. Quan, '"NP-Hard Problems of Learning from Examples," Fifth International Conference on Fuzzy Systems and Knowledge Discovery, vol. 2, pp. 182-186, 2018.
[58] P.A. Vikhar, '"Evolutionary algorithms: A critical review and its future prospects," International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), pp. 261-265. 2016.
[59] J. Holland, '"Adaptation in Natural and Artificial Systems,", 1992.
[60] L. Chambers, '"The Practical Handbook of Genetic Algorithms,", 1995.
[61] J. Lee, J. Kim, D. Kang, N. Kim and S. Jung, '"Cloud service broker portal: Main entry point for multi-cloud service providers and consumers," 16th International Conference on Advanced Communication Technology, pp. 1108-1112, 2015.
[62] S. Liu and Z. Li, '"A modified genetic algorithm for community detection in complex networks," International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET), pp. 1-3, 2017.
[63] R. Abuabara, '"Tabu Search para el problema de ubicación del Controlador en Software Defined Networks,", pp. 86, 2019.

[64] University of Adelaide, "'The Internet Topology Zoo,".
[65] V. Hegde, T.S. Aswathi and R. Sidharth, "'Student residential distance calculation using Haversine formulation and visualization through GoogleMap for admission analysis," IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pp. 1-5, 2016.
[66] J. Jiang, H. Huang, J. Liao and S. Chen, "'Extending Dijkstra's shortest path algorithm for software defined networking," The 16th Asia-Pacific Network Operations and Management Symposium, pp. 1-4, 2014.