Maestría en Ingeniería Civil

Optimización del diseño de una red de suministro de agua para un proyecto de expansión en la sabana de Bogotá

Henry Omar Peñaloza Mantilla

Bogotá, D.C., 30 de enero de 2020

Escuela Colombiana de Ingeniería
Julio Garavito

Optimización del diseño de una red de suministro de agua para un proyecto de expansión en la sabana de Bogotá

Tesis para optar al título de magíster en Ingeniería Civil, con énfasis en Recursos Hidráulicos y Medio Ambiente

German Ricardo Santos Granados

Director

Bogotá, D.C., 30 de enero de 2020

Escuela Colombiana de Ingeniería
Julio Garavito

La tesis de maestría titulada “Optimización del diseño de una red de suministro de agua para un proyecto de expansión en la sabana de Bogotá”, presentada por Henry Omar Peñaloza Mantilla, cumple con los requisitos establecidos para optar al título de Magíster en Ingeniería Civil con énfasis en Recursos Hidráulicos y Medio Ambiente

Director de la tesis

German Ricardo Santos Granados

Jurado

Andres Humberto Otálora Carmona

Jurado

Freddy Santiago Duarte Prieto

Bogotá, D.C., 30 de enero de 2020

Dedicatoria

A Zulay, por su amor y apoyo aún en los peores momentos.

A Damián, por ser el motor de mi vida.

Agradecimientos

Al Doctor German Ricardo Santos Granados, por guiarme en la elaboración de este proyecto.

Al Ingeniero Félix Ernesto Orduz Grimaldo, por su invaluable colaboración durante el desarrollo de esta tesis.

Al Ingeniero Jaime Alberto Vargas Rey, por creer en mí.

Resumen

La necesidad de hacer eficientes y económicamente viables las grandes inversiones necesarias para la construcción de redes de abastecimiento de agua, hace que se preste especial atención al diseño de este tipo de sistemas. Concebir soluciones económicamente optimas y que garanticen un adecuado funcionamiento de los sistemas de acueducto es uno de los grandes retos que, durante las últimas tres décadas, han abordado numerosas investigaciones, las cuales han centrado sus esfuerzos en desarrollar metodologías de optimización que pudieran aplicarse a su diseño.

En general, el diseño de redes de distribución de agua es un problema que por su complejidad se clasifica como NP-hard (Yates, Templeman, & Boffey, 1984); es decir que es un problema para el cual no se conoce ningún método determinístico que permita solucionarlo en un tiempo razonable. El problema radica esencialmente en la elección de los diámetros como variables de decisión. Las restricciones son funciones implícitas de estos mismos diámetros.

Este trabajo se centra fundamentalmente en el dimensionamiento hidráulico de redes de distribución de agua, de modo que se cumplan una serie de requerimientos de servicio, entre los que se incluyen las condiciones mínimas y máximas de presión y velocidad en la red. Se desarrollaron en Python los algoritmos evolutivos conocidos como Algoritmo Genético (AG) y Particle Swarm Optimization (PSO), utilizando principalmente las librerías DEAP y EPANETTOOLS.

En esta tesis se comparan los resultados obtenidos por los mencionados algoritmos evolutivos al aplicarlos en el diseño de una red real construida en la sabana de Bogotá y en una red en la que se han evaluado diversos algoritmos de optimización, que han sido ampliamente documentados en la literatura, la red de Hanoi.

Las técnicas heurísticas desarrolladas, aunque no garantizan encontrar la solución óptima, contienen estrategias que permiten alcanzar los objetivos y encontrar soluciones cercanas a

la óptima. En ocasiones es preferible encontrar al menos una solución aproximada, pero viable, aunque algunos pocos parámetros puedan no satisfacer estrictamente una restricción.

Índice general

Índice de tablas

Índice de figuras

Índice de anexos

Introducción

América Latina y el Caribe cuenta con un poco más de 620 millones de habitantes y es la región más urbanizada del mundo en vías desarrollo con un 80% de su población viviendo en ciudades. El número de ciudades con más de 500.000 habitantes se ha multiplicado por casi cinco (de 28 a 131) en los últimos cincuenta años y más de la mitad de la población urbana (55%) reside en ellas (Rojas, 2014).

El desarrollo de las ciudades trae consigo un crecimiento de la población a la cual hay que garantizarle el acceso al agua potable. Este problema es de vital importancia en la región donde, aunque durante los últimos años ha aumentado la cobertura del servicio de acueducto, aún una buena parte de la población no tiene acceso al agua potable y sus recursos económicos son muy limitados.

Los costos asociados a la construcción, ampliación y mantenimiento de los sistemas de distribución de agua son relativamente altos y representan una parte significativa del presupuesto de los países latinoamericanos.

Durante el periodo de tiempo comprendido entre los años 2002 a 2009, en países como Colombia la inversión en el sector del agua potable y el saneamiento básico superó los 13,4 mil millones de dólares, equivalentes a una inversión anual de casi 1,7 mil millones. En el cuatrienio 2006-2009, la inversión fue superior en un 17% respecto al 2002-2005. Los recursos asignados al sector alcanzaron 1,9 mil millones en 2009 y casi dos mil millones en 2010 (Rojas, 2014).

Para cumplir con el objetivo de alcanzar una cobertura del 100% en el servicio de agua potable, los sistemas de acueducto deben ser capaces de hacer frente a las continuas y cambiantes necesidades de la población a la que atienden, adaptándose a las crecientes demandas de consumo y expansión de las zonas de suministro (Tolson, Maier, Simpson, & Lence, 2004).

En la actualidad, existen diversos métodos para realizar diseños de redes de distribución de agua potable. Muchos de ellos dependen en gran parte de la experiencia del diseñador en la implementación de técnicas empíricas para la selección de los diámetros de las tuberías. Esto resulta muchas veces en diseños que funcionan desde el punto de vista hidráulico. Sin embargo, al no ser óptimos desde el punto de vista económico porque tienden a generar diámetros grandes con altas presiones, incrementan los costos de las tuberías y sus accesorios, y aumentan la cantidad de mantenimientos causados por las altas presiones que originan pérdidas de agua y daños en las líneas de conducción.

Debido a lo anterior, en Colombia el Ministerio de Vivienda Ciudad y Territorio (2017) establece que los diseños hidráulicos se deben realizar bajo criterios de diseño exhaustivo u optimizado y que la elección de un determinado diámetro para un tramo de tubería debe basarse en un estudio comparativo técnico – económico, mediante técnicas de optimización que hagan que el costo de la obra objeto de diseño sea mínimo.

La presente investigación desarrolla e implementa una metodología que permite incluir criterios de optimización económicos en el diseño de redes de distribución de agua potable, para esto se aplican dos técnicas de optimización, Algoritmos Genéticos (AG) y Optimización de enjambre de partículas (PSO, Particle Swarm Optimization), al "Diseño para el abastecimiento de agua potable (sectorización) del sistema de acueducto para grandes desarrollos de vivienda e industriales del municipio de Mosquera Departamento Cundinamarca" elaborado por Integrita S.A.S. (2014) y al diseño de la red Hanoi (Fujiwara & Khang, 1990).

Este trabajo de grado se desarrolla en 4 capítulos, a saber: Capítulo I - Estado del arte en la optimización del diseño de redes de distribución de agua, Capítulo II – Metodología, Capítulo III - Resultados y contribución y Capitulo IV - Aplicación de los Algoritmos Genético y PSO a un ejemplo reportado por la literatura, para finalizar con algunas conclusiones y recomendaciones producto del estudio.

Capítulo I

Estado del arte en la optimización del diseño de redes de distribución de agua

Para el diseño de redes de distribución de agua, intuitivamente los ingenieros han usado por años el método de prueba y error, basado en el sentido común y la experiencia para intentar mejorar progresivamente sus soluciones. La introducción de la computación dirigida al diseño en ingeniería y el desarrollo de varias técnicas de optimización han permitido sustanciales mejoras en el proceso de búsqueda de buenas soluciones (Montalvo, 2008). Una buena solución es una solución de diseño que no exceda sustancialmente la solución óptima que se obtiene de las simulaciones (D. Mora, Iglesias, Martinez, & Fuertes, 2013).

Los modelos de optimización se han formulado y resuelto para diseñar sistemas de distribución de agua desde la década de 1970. Existe una gran cantidad de literatura sobre el diseño y la optimización de redes de distribución de agua utilizando la programación lineal (PL), programación no lineal (PNL), técnicas de enumeración (TE), métodos heurísticos (MH) y técnicas evolutivas (TE).

Para lograr el diseño óptimo de redes de distribución de agua, la literatura ha reportado el uso de técnicas destacadas por su efectividad, tales como: algoritmos genéticos (AG) (Simpson, Dandy, & Murphy, 1994); (Dandy, Simpson, & Murphy, 1996); (Savic & Walters, 1997); (Vairavamoorthy & Ali, 2000); (Kadu, Gupta, & Bhave, 2008); algoritmo de colonia de hormigas (Maier et al., 2003); algoritmo de salto de rana barajado (Eusuff & Lansey, 2003); búsqueda de armonía (Geem, 2006); optimización de enjambre de partículas (Suribabu & Neelakantan, 2006); autómatas celulares (Keedwell & Khu, 2006); búsqueda tabú (Sung, Lin, Lin, & Liu, 2007); evolución diferencial (Suribabu, 2010); (Vasan & Simonovic, 2010); y optimización de apareamiento de abejas melíferas (Mohan & Babu, 2010), entre otras.

A su vez, muchos otros trabajos han evaluado el rendimiento de las técnicas de optimización mencionadas, utilizando redes de referencia pequeñas o medianas propuestas también en la

literatura, pero pocos de ellos han probado estos métodos con redes reales a gran escala (Baños, Gil, Reca, & Montoya, 2010).

Simpson y otros (1994) aplicaron las técnicas de algoritmo genético, numeración completa y programación no lineal en la optimización de la red de túneles de Nueva York propuesta por Morgan & Goulter (1985). Al revisar los resultados obtenidos se evidenció que la técnica del algoritmo genético encuentra el óptimo global en relativamente pocas evaluaciones en comparación con el tamaño del espacio de búsqueda.

Iglesias, Mora, Lopez & García (2009) aplicaron distintas técnicas evolutivas, como: algoritmos genéticos, optimización de enjambre de partículas, búsqueda de armonía y algoritmo de salto de rana barajado, en la optimización de la red Hanoi (Fujiwara & Khang, 1990) y la red de túneles de Nueva York (Morgan & Goulter, 1985); la comparación de los resultados obtenidos no le permitió a los autores concluir cuál de los modelos propuestos es el mejor, aunque fue claro que, en mayor o menor medida, todas las técnicas utilizadas son válidas para el diseño de redes de agua.

Baños y otros (2010) desarrollaron un nuevo algoritmo memético para el diseño óptimo de redes de distribución de agua, el cual se aplicó a tres redes de distribución y se comparó con otros cinco métodos conocidos en la literatura, a saber, recocido simulado, recocido simulado mixto, búsqueda de tabú, búsqueda por dispersión, algoritmos genéticos y programación de enteros lineales binarios. Los resultados obtenidos mostraron que el algoritmo memético se desempeñó mejor que los demás métodos estudiados.

De otra parte, Mora (2012) en su tesis doctoral realizó un análisis estadístico, no sólo de los resultados obtenidos al aplicar las siguientes técnicas heurísticas de optimización: algoritmos genéticos, recocido simulado, optimización de enjambre de partículas, algoritmo de salto de rana barajado y búsqueda de armonía, sino también de la influencia que ejerce el ajuste de los distintos parámetros de cada técnica en el buen hacer de la misma.

Zheng, Zecchin, Simpson, & Lambert (2014) desarrollaron un algoritmo genético basado en la mutación (CMBGA) para la optimización de una red de tuberías. Este CMBGA difiere de la optimización del algoritmo genético clásico (AG) en que no utiliza el operador de cruce; en su lugar, solo utiliza la selección y un operador propuesto de mutación progresiva. Para investigar la efectividad del CMBGA propuesto, este, en conjunto con otras cuatro variantes de AG se aplicó en dos estudios de caso. Los resultados mostraron que el CMBGA propuesto exhibe una mejora considerable con respecto a los valores obtenidos de la aplicación de los AG considerados y un rendimiento similar al reportado en otros resultados publicados con anterioridad. Dos grandes ventajas del CMBGA son su simplicidad y el hecho de que requiere el ajuste de menos parámetros en comparación con las otras variantes de AG.

Posteriormente, Mora, Iglesias, Martinez & Ballesteros (2015) presentaron una metodología para comparar algoritmos basada en una tasa de eficiencia (E), la cual se aplicó al problema de dimensionamiento de tuberías de cuatro redes de referencia de tamaño mediano (Hanoi, New York Tunnel, GoYang y R-9 Joao Pessoa). La tasa de eficiencia numéricamente determina el rendimiento de un algoritmo dado mientras considera la calidad de la solución obtenida y el esfuerzo computacional requerido. De la amplia gama de algoritmos evolutivos disponibles, los autores seleccionaron cuatro algoritmos para implementar la metodología: algoritmo pseudogenético (PGA), optimización de enjambre de partículas (PSO), búsqueda de armonía (HS) y un algoritmo de salto de rana barajado (SFLA). Después de más de 500,000 simulaciones, los autores realizaron un análisis estadístico basado en los parámetros específicos que cada algoritmo requiere para operar, y finalmente, se analizó E para cada red y algoritmo. La medida de eficiencia indicó que el PGA es el algoritmo más eficiente para problemas de mayor complejidad y que el HS es el algoritmo más eficiente para problemas menos complejos. Sin embargo, la principal contribución de este trabajo fue que el índice de eficiencia propuesto proporciona una estrategia neutral para comparar algoritmos de optimización y puede ser útil en el futuro para seleccionar el algoritmo más adecuado para diferentes tipos de problemas de optimización.

Dardani & Jones (2018), evaluaron tres algoritmos de minimización de costos en seis redes de agua de escala moderada e impulsadas por gravedad. Dos de los algoritmos evaluados, un algoritmo de retroceso y un algoritmo genético, utilizan un conjunto de diámetros de tubería discretos, mientras que un nuevo algoritmo basado en cálculo produce una solución de diámetro continuo que se asigna a un conjunto de diámetros discretos. Este estudio resaltó las ventajas y debilidades de cada método de diseño de redes de agua impulsada por gravedad, incluida la cercanía al óptimo global, la capacidad de reducir el espacio de la solución eliminando los candidatos poco viables y subóptimos, sin perder el óptimo global y sin aumentar el tiempo de ejecución del algoritmo.

A nivel regional también se han desarrollado diversos trabajos que formulan y aplican técnicas de optimización al diseño de redes de distribución de agua, algunas de las técnicas aplicadas son: recocido simulado (Cunha & Sousa, 1999), (Sanvicente Sanchez & Solís, 2003), (Méndez, 2014); búsqueda tabú (Cunha & Ribeiro, 2004); y algoritmo genético (Rincón, 2006), (Rodríguez, Fuentes, Jiménez, & Cruz, 2006), (Pino et al., 2017).

Pereyra, Pandolfi & Villagra (2017) aplicaron cuatro técnicas de optimización derivadas de los algoritmos genéticos: crossover elitism population, half uniform crossover combination, cataclysm mutation (CHC) y el algoritmo genético canónico, en dos redes clásicas de distribución de aguas: red Alperovits & Shamir (1977) y red de túneles de Nueva York (Morgan & Goulter, 1985). Con la aplicación de las cuatro técnicas se obtuvieron en general resultados satisfactorios y de gran calidad.

En Colombia, Nárvaez & Galeano (2002) presentaron una ecuación de costos para redes que manejan líquidos, que en conjunto con el modelo y el método de solución del problema de mecánica de fluidos, constituyen la base fundamental sobre la cual trabajan la programación lineal, la programación no lineal y los algoritmos genéticos.

Lopez, Cuero, Díaz, Páez & Saldarriaga (2013), implementaron diseños obtenidos en la etapa final de la metodología de superficie de uso óptimo de potencia (OPUS) como un "inicio en

caliente" para las metodologías metaheurísticas típicamente utilizadas en el diseño de redes de agua potable, tales como: algoritmos genéticos (AG), búsqueda de armonía (HS) y recocido simulado (SA). Esto se propuso con el objetivo de mejorar los resultados ya alcanzados con criterios hidráulicos. La metodología propuesta fue probada en 3 redes de referencia: Hanoi (Fujiwara & Khang, 1990), Balerma (Reca & Martínez, 2006) y Taichung (Sung et al., 2007), además de una cuarta red hipotética conocida como R28 creada con fines de investigación en el Centro de Investigación de Acueducto y Alcantarillado (CIACUA) de la Universidad de Los Andes en Bogotá, Colombia, y arrojó resultados muy cercanos a los registros globales reportados en la literatura, pero con una diferencia poco significativa en relación con la metodología OPUS. Además, el número de iteraciones aumentó sustancialmente con respecto a la metodología de OPUS, lo que les permitió a los autores concluir que la mejora de los diseños hidráulicos desarrollados con la metodología OPUS requiere un gran esfuerzo computacional adicional para una reducción mínima en los costos.

Capítulo II
Metodología

2.1. Preguntas de investigación

2.1.1. General

¿Cómo elaborar diseños de redes de acueducto óptimos desde el punto de vista económico utilizando EPANET y librerías de Python?

2.1.2. Especificas

¿Cómo establecer los costos generales de construcción para una red de distribución de agua a nivel local?

¿Cuáles son las restricciones hidráulicas y comerciales que deben aplicarse a nivel local para el diseño de una red de distribución de agua potable?

¿Cómo se implementa una técnica de optimización utilizando epanet en python?

¿Cuán efectiva es la aplicación de técnicas de optimización en la solución del problema de diseño de redes de distribución de agua potable?

2.2. Objetivos

2.2.1. Objetivo general

Desarrollar e implementar una metodología que permita incluir criterios de optimización económicos en el diseño de redes de distribución de agua potable utilizando epanet y librerías desarrolladas en Python

2.2.2. Objetivos específicos

Deducir la ecuación general de costos para una red de distribución de agua construida a nivel local.

Establecer las restricciones hidráulicas y comerciales aplicables en el diseño de redes de distribución de agua potable a nivel local.

Aplicar en el diseño de una red suministro de agua, construida para un proyecto de expansión tipo en la sabana de Bogotá, dos técnicas de optimización reportadas por la literatura implementadas en python.

Comparar objetivamente los resultados obtenidos al aplicar las técnicas de optimización propuestas con los resultados del diseño efectivamente utilizado en la construcción de la red suministro de agua del proyecto de expansión tipo seleccionado en la sabana de Bogotá.

2.3. Tipo de investigación

En el presente trabajo de grado se desarrollan prácticas entendidas como investigación aplicada, ya que se caracterizan porque buscan la aplicación o utilización de conocimientos adquiridos, a la vez que se adquieren otros después de implementar y sistematizar la práctica basada en investigación.

Estas prácticas son experiencias de investigación cuyo propósito es resolver o mejorar una situación específica o particular, para comprobar un método o modelo mediante la aplicación innovadora y creativa de una propuesta de intervención (Vargas Cordero, 2009).

2.4. Técnicas heurísticas aplicadas al diseño de redes de acueducto

Cuando se requiere resolver problemas de búsqueda tan grandes como los relacionados con el diseño de una red de distribución de agua, las técnicas clásicas de búsqueda y optimización resultan en general insuficientes, es aquí donde las técnicas heurísticas juegan su papel.

Una heurística es una técnica que busca buenas soluciones a un costo computacional razonable, aunque sin garantizar la factibilidad de las mismas. En algunos casos, ni siquiera se puede determinar qué tan cerca del óptimo se encuentra una solución factible en particular (Reed, Toombs, & Barricelli, 1967).

Este trabajo aplica dos metodologías: Algoritmos Genéticos (AG) y Particle Swarm Optimization (PSO), al diseño óptimo de redes de agua, estableciendo criterios de comparación entre las mismas, a fin de determinar las ventajas e inconvenientes que puedan asociarse a cada una de ellas.

2.4.1. Algoritmos genéticos

Un Algoritmo Genético es una técnica de optimización numérica que se basa en las leyes de la evolución de Darwin. Su forma de trabajo trata de simular la evolución de una población de individuos, sometida a acciones aleatorias semejantes a las que actúan en la evolución biológica (reproducción, cruce y mutación), así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles son los menos aptos, que son descartados (Mora, 2012).

Mediante la aplicación del Algoritmos Genéticos es posible determinar la red de costo mínimo (individuo mejor adaptado) tras una serie de iteraciones (generaciones), gracias a unas reglas de transición (operadores genéticos) encargadas de elegir las

mejores soluciones (costos más bajos) según el valor de la función objetivo (costo) de cada una de ellas, partiendo de un conjunto de diseños iniciales establecidos al azar (Montesinos, García-Guzmán, & Ayuso, 1997).

La aplicación diseñada en este trabajo usa como parámetro de entrada al modelo una lista discreta de diámetros comerciales. El funcionamiento del Algoritmo Genético aplicado al diseño de redes de agua viene representado en forma de diagrama de flujo en la Figura 1.

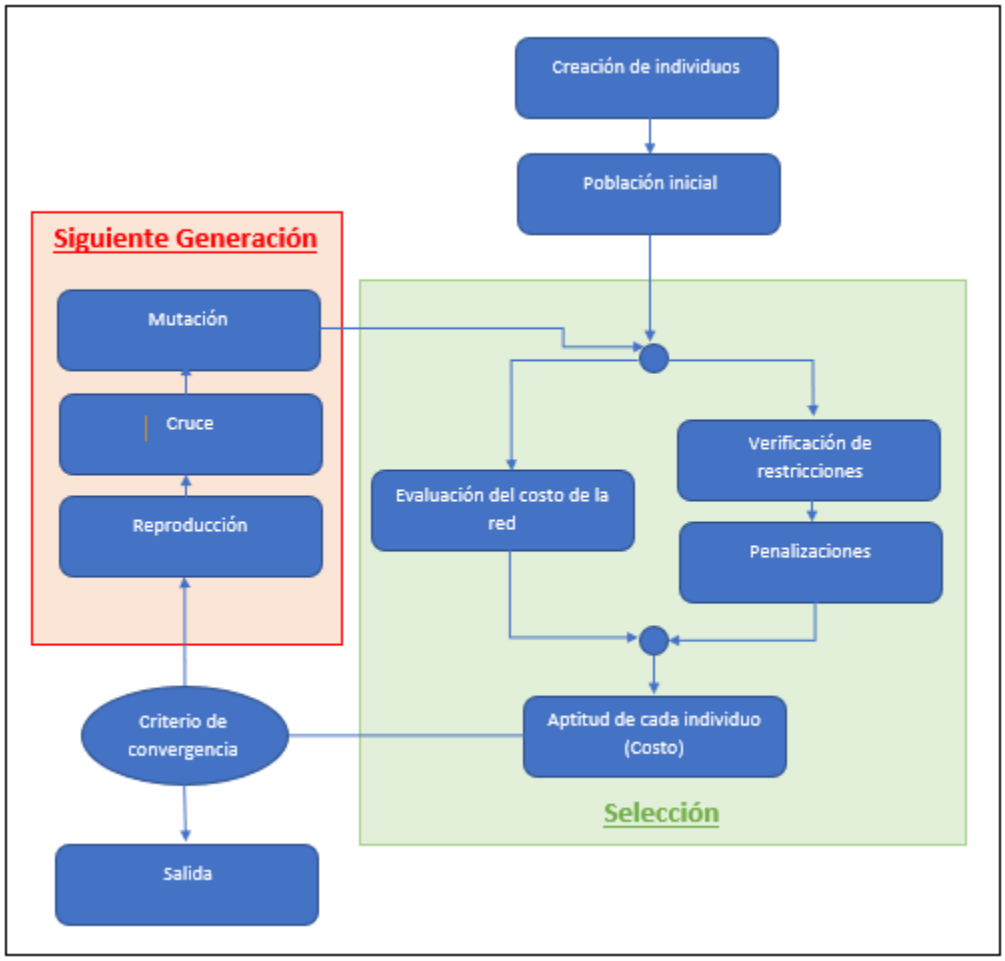


Figura 1. Algoritmo Genético aplicado al diseño óptimo de redes de acueducto. Fuente: Elaboración propia.

La idea básica es generar un conjunto de posibles soluciones, cada una de las cuales es llamada individuo. El conjunto de todas estas posibles soluciones es lo que se conoce como población del algoritmo. La población es sometida a la aplicación de tres “operadores genéticos”: reproducción, cruce y mutación.

Antes de iniciar el cálculo, es fundamental la definición de los siguientes cuatro parámetros: tamaño de la población, probabilidad de cruce, probabilidad de mutación, y número de iteraciones que realizará el algoritmo, que en este caso es la condición de convergencia que se impone al algoritmo.

El cálculo comienza con la creación de un individuo a partir de la escogencia aleatoria de serie de cromosomas (diámetros comerciales de tubería), cada uno de estos individuos es en sí una posible solución del problema de optimización. Esta primera generación es totalmente aleatoria.

Los individuos conforman la población. En el problema de diseño planteado, la aptitud de un determinado individuo viene dada por el costo asociado a la construcción de la red de agua.

El software desarrollado en Python utiliza la librería DEAP para construir un Algoritmo genético capaz de trabajar en conjunto con el software de simulación hidráulico EPANET y su librería, EPANETTOOLS. Cada uno de los individuos proporcionados por el Algoritmo es simulado en EPANET, con el objetivo de verificar que la solución dada cumple todas las restricciones impuestas.

Posteriormente, los individuos se reproducen. Los mejores individuos tienen mayor probabilidad de reproducirse, mientras que los peores tienen menor probabilidad de reproducirse.

En el Algoritmo diseñado se utiliza el método de reproducción constante, el cual consistente en asignar a cada cadena de la población una probabilidad de reproducción, que dependerá directamente de la aptitud de cada cadena.

Los individuos de la nueva generación son emparejados, produciéndose un intercambio de material genético. Los operadores genéticos aplicados son dos: cruce y mutación.

El proceso de cruce escoge de modo aleatorio ciertos individuos de la población creada y realiza un cambio de distintos genes (diámetros) a partir de un cierto gen de cruce. Se genera entonces un nuevo individuo que es combinación de otros dos.

El proceso de mutación se aplica a la población obtenida tras los procesos de reproducción y cruce, para esto se establece una probabilidad de mutación, se examina cada gen o diámetro de tubería en la red, y si un número aleatorio generado por el Algoritmo está por debajo de esa probabilidad se produce un cambio en el valor del diámetro.

El ciclo se repite una y otra vez hasta que se cumpla la condición de convergencia. En la aplicación diseñada, el Algoritmo repite el ciclo hasta que se produzca un cierto número de repeticiones sin ninguna mejora en el mejor individuo de la población. Este número es lo suficientemente alto como para evitar una convergencia temprana en un mínimo local, pero no tanto como para ralentizar excesivamente la convergencia del algoritmo.

2.4.2. Optimización por Enjambre de Partículas PSO (Particle Swarm Optimization)

La Optimización por Enjambre de Partículas (conocida como PSO, por sus siglas en inglés, Particle Swarm Optimization) es una técnica de optimización/búsqueda propuesta por Kennedy & Eberhart (1995) que se basa en el comportamiento social

que tienen los enjambres (poblaciones) de partículas cuando se desplazan a lugares desconocidos. Los “swarms” o enjambres son poblaciones de partículas que representan soluciones potenciales de un problema determinado. La búsqueda hacia nuevas soluciones es guiada por lo que podríamos denominar presión social, es decir, está basada en el conocimiento que tienen unos individuos de otros. Todas las partículas pertenecientes a un “swarm” se comunican entre sí para direccionar la búsqueda, emulando las técnicas de exploración que emplean los insectos o pájaros en la naturaleza (Mora, 2012).

Cada una de estas partículas lleva asociado un “fitness”, que indica la aptitud de la misma, así como un vector “fitness best”, que hace referencia a la mejor posición en el espacio de soluciones en la que ha estado ese individuo en particular.

En el proceso evolutivo en PSO cada partícula desarrolla un comportamiento social, adecuando su movimiento a la búsqueda de un objetivo. Cada partícula se desplaza en una dirección específica, pero al estar comunicadas entre sí son conocedoras de que partícula se encuentra en mejor posición. De este modo, el resto de partículas se desplazan siguiendo a la que este mejor posicionada, con una velocidad que dependerá de su posición actual. Cada una de estas partículas investiga el espacio de búsqueda desde su nueva posición local, y el proceso se repite hasta que la bandada converge hacia la mejor solución.

En general, las partículas aprenden tanto de su propia experiencia (búsqueda local) cómo de la experiencia del resto de la bandada (búsqueda global).

Inicialmente el algoritmo PSO fue propuesto para la optimización en espacios continuos, siendo escaso hasta el momento el trabajo aportado para su aplicación en espacios discretos (Daniel Mora, 2012). Para conseguir el diseño óptimo de redes de agua, dado que el objetivo es determinar los diámetros de las tuberías, este trabajo utiliza la optimización en espacios discretos, pues es más práctico obtener una

solución de diámetros comerciales directamente en lugar de tener que realizar una normalización posterior.

A continuación, en la figura 2 se presenta el proceso de optimización que sigue el algoritmo PSO propuesto:

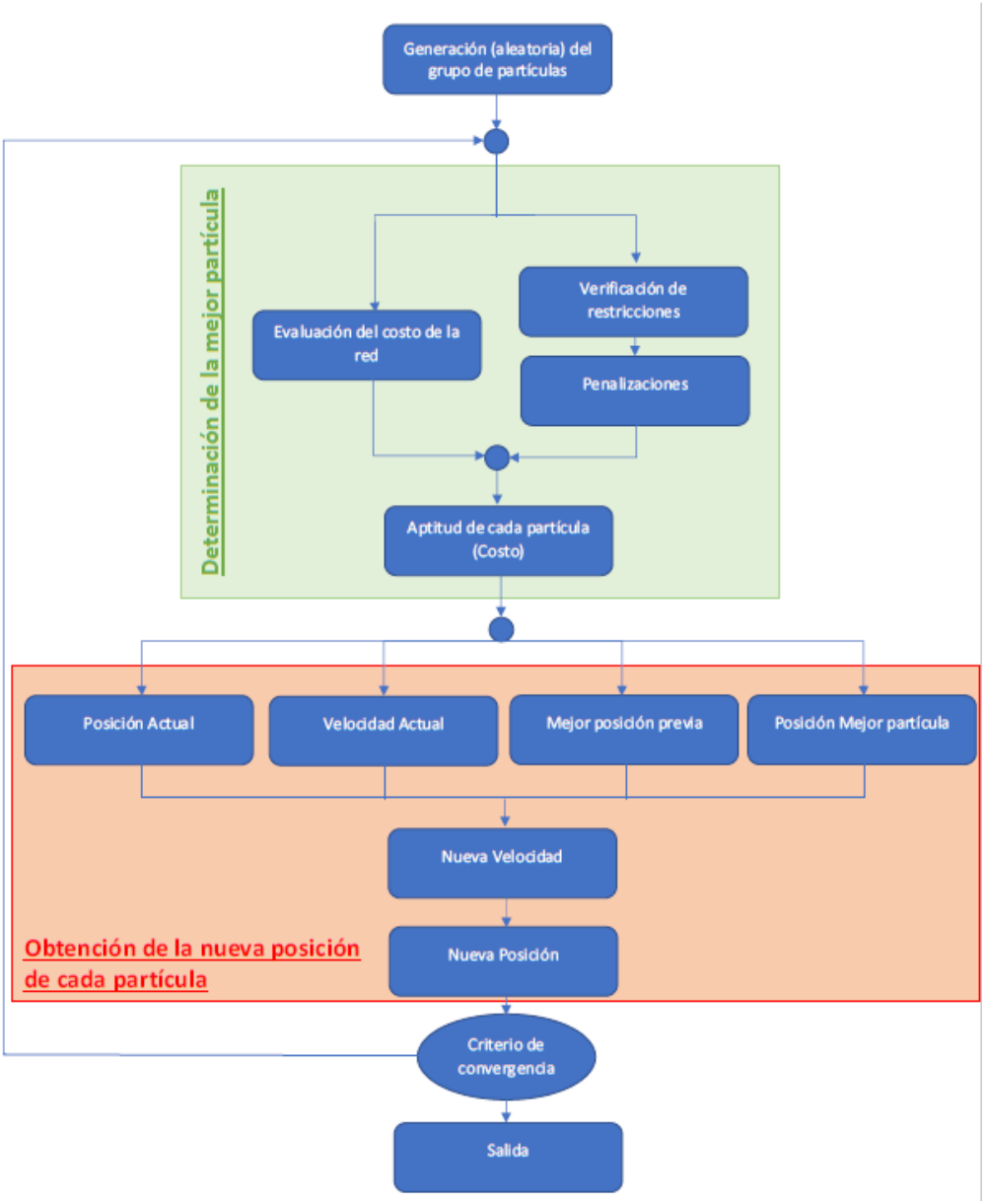


Figura 2. Particle Swarm Optimization (PSO) aplicado al diseño óptimo de redes de acueducto.

Fuente: Elaboración propia.

El proceso comienza con la generación aleatoria de un grupo de partículas, cada una de las cuales representa una solución del problema.

La posición actual de una partícula está determinada por su aptitud (costo de la red). La mejor de todas ellas actúa como una referencia para las demás, de modo que el resto desarrolla un comportamiento de aprendizaje.

En cada ciclo del algoritmo todas las partículas cambian su posición, intentando alcanzar la posición de la mejor partícula en cada instante (best). La velocidad de vuelo de cada partícula está relacionada con lo cerca o lejos que ésta se encuentra de la partícula líder.

El algoritmo se ejecuta hasta que se produce la condición de convergencia. La condición de convergencia fijada para PSO es la misma que la que se ha utilizado en el Algoritmo Genético; es decir, el algoritmo finaliza su tarea cuando se cuenta con un número determinado de iteraciones sin obtener mejora alguna en la solución.

La velocidad máxima a la que pueden desplazarse las partículas es un parámetro de vital importancia en el algoritmo PSO, puesto que este parámetro determina la zona de búsqueda de soluciones en la que se centra el algoritmo. De este modo, si la velocidad máxima tiene un valor excesivamente alto es muy posible que las partículas pasen de largo determinados espacios de solución. Al contrario, si la velocidad máxima es demasiado pequeña, las partículas exploran suficientemente el espacio de soluciones más allá de la zona local en la que se mueven, es decir, que lo más probable es que queden atrapados en un mínimo local y siendo incapaces de salir.

Normalmente se consideran valores máximos de velocidad que oscilan entre el 10 y el 20% del valor de la variable que se pretende optimizar. Si extrapolamos este dato al diseño de redes de agua, donde la variable de decisión es el diámetro de tubería, se

tiene que, para una margen de 10 diámetros, se tendría una velocidad máxima que permitirá un máximo salto de 2 diámetros entre iteración e iteración.

2.5. Caso de estudio

El crecimiento poblacional experimentado por los municipios ubicados en la Sabana de Bogotá, en especial el ocurrido en el Municipio de Mosquera - Cundinamarca, condujo a una mayor demanda de agua potable y a la consecuente ampliación de la red de distribución que permitió dar servicio a los nuevos proyectos urbanísticos e industriales que se desarrollaron como parte del crecimiento y la expansión que el municipio proyectó a corto y mediano plazo en zonas en expansión urbana y a través de Planes Parciales.

Para conseguir lo anterior, el gestor del servicio de acueducto en el Municipio de Mosquera, contrató en el año 2014 un estudio que se denominó: "Diseño para el abastecimiento de agua potable (sectorización) del sistema de acueducto para grandes desarrollos de vivienda e industriales del municipio de Mosquera Departamento Cundinamarca" (Integrita S.A.S., 2014) y que permitió estructurar un sistema abastecimiento de agua potable para los desarrollos industriales y de vivienda relacionados a continuación: Parque Industrial Zona Franca de Occidente, Parque Industrial San Jorge, Parque Industrial Santo Domingo, proyectos de vivienda como: Ciudad Sabana, Plan Parcial San Luis, Senderos de Siete Trojes, Plan Parcial Salesianos, Hacienda Alcalá, El Novillero, y Plan Parcial San José. El estudio dispone de un modelo hidráulico de la red que permite evaluar su comportamiento futuro, estimando las demandas para el año horizonte del proyecto (2024).

La red está compuesta por un total de 34 líneas, de las cuales 1 es una bomba. Las tuberías permiten la conexión de la fuente con los 33 nudos de la red, de los que 1 es un depósito. Un total de 16 nudos tienen un consumo aplicado.

El modelo de la red en formato .INP para su tratamiento en EPANET se puede observar en la figura 3.

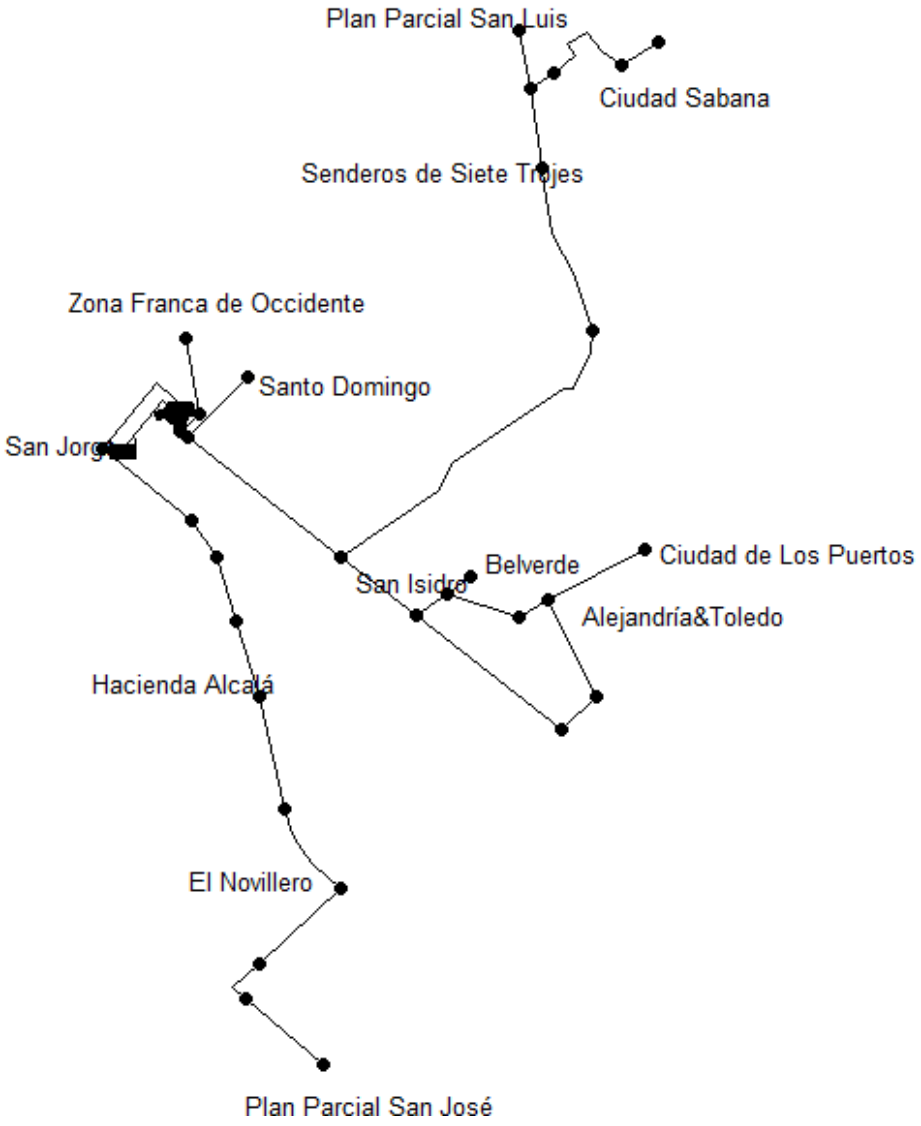


Figura 3. Esquema de conexión de nodos y tuberías Modelo hidráulico.
Fuente: Integrita S.A.S., 2014

Un resumen de los resultados de la modelación ejecutada para el diseño inicial se puede observar en las tablas 1 y 2.

Tabla 1. Resultados de la modelación (nodos) Modelo Año 2024

ID Nodo	Cota	Cota Rel.	Demanda	Altura	Presión
	msnm	M	LPS	M	Mca
3	2545,00	45,00	0,00	79,70	34,70
4	2545,00	45,00	0,00	79,44	34,44
5	2544,50	44,50	0,00	78,67	34,17
6	2544,03	44,03	0,00	78,61	34,58
7	2543,15	43,15	0,00	78,59	35,44
8	2544,00	44,00	3,90	78,00	34,00
9	2544,00	44,00	3,90	77,97	33,97
10	2543,80	43,80	3,60	77,63	33,83
11	2543,80	43,80	8,50	77,62	33,82
12	2544,20	44,20	9,30	77,14	32,94
13	2545,00	45,00	13,40	79,25	34,25
14	2545,90	45,90	0,00	76,09	30,19
15	2547,00	47,00	3,10	73,28	26,28
16	2548,00	48,00	0,00	71,99	23,99
17	2548,00	48,00	7,00	71,43	23,43
18	2548,20	48,20	6,00	70,64	22,44
19	2548,30	48,30	6,00	70,20	21,90
20	2548,20	48,20	44,20	71,29	23,09
21	2545,00	45,00	0,00	79,57	34,57
23	2545,00	45,00	8,60	79,44	34,44
24	2545,00	45,00	11,30	78,85	33,85
25	2544,80	44,80	0,00	78,04	33,24
26	2544,60	44,60	0,00	77,23	32,63
27	2543,90	43,90	0,00	76,25	32,35
28	2543,20	43,20	14,50	75,71	32,51
29	2542,50	42,50	0,00	75,35	32,85
30	2541,80	41,80	16,70	74,99	33,19
31	2541,70	41,70	0,00	74,79	33,09
32	2541,70	41,70	0,00	74,59	32,89
33	2541,60	41,60	42,50	74,40	32,80
22	2543,40	43,40	0,00	78,58	35,18
35	2545,00	45,00	0,00	79,65	34,65
Embalse 1	2549,10	49,10	-55,81	49,10	0,00
Depósito 2	2545,00	45,00	-146,69	49,00	4,00

Tabla 2. Resultados de la modelación (tuberías) Modelo Año 2024

#	Nodos		Long.	Diámetro		Caudal	Velocidad	Pérd. Unit.	Factor de Fricción
				Nominal	Interno				
	De	A	m	Pulg	Mm	LPS	m/s	m/km	
1	1	2	322,00	18"	422,04	55,81	0,40	0,31	0,016
3	3	35	85,20	20"	468,94	108,90	0,63	0,63	0,015
6	4	5	535,50	16“	375,16	95,50	0,86	1,45	0,014
7	5	7	312,80	16"	375,16	29,20	0,26	0,17	0,018

#	Nodos		Long.	Diámetro		Caudal	Velocidad	Pérd. Unit.	Factor de Fricción
				Nominal	Interno				
	De	A	m	Pulg	Mm	LPS	m/s	m/km	
8	6	8	597,30	16"	375,16	13,07	0,12	0,04	0,021
9	6	8	150,60	6"	155,32	16,13	0,85	4,05	0,017
10	8	9	100,00	6"	155,32	3,90	0,21	0,32	0,023
11	8	10	300,00	6"	155,32	8,33	0,44	1,24	0,020
12	11	10	24,70	6"	155,32	-4,73	0,25	0,45	0,022
13	11	12	317,40	6"	155,32	9,30	0,49	1,51	0,019
14	22	11	347,80	6"	155,32	13,07	0,69	2,77	0,018
15	5	14	1156,00	12"	298,95	66,30	0,94	2,23	0,015
16	14	15	554,60	10"	252,07	66,30	1,33	5,07	0,014
17	15	16	277,80	10"	252,07	63,20	1,27	4,65	0,014
18	16	17	102,90	6"	155,32	19,00	1,00	5,43	0,016
19	17	18	332,90	6"	155,32	12,00	0,63	2,38	0,018
20	18	19	100,00	4"	105,52	6,00	0,69	4,38	0,019
21	16	20	100,00	8"	202,17	44,20	1,38	7,03	0,015
22	4	13	239,40	8"	202,17	13,40	0,42	0,82	0,019
23	24	25	300,00	12“	298,95	73,70	1,05	2,70	0,014
24	25	26	300,00	12"	298,95	73,70	1,05	2,70	0,014
25	26	27	362,50	12"	298,95	73,70	1,05	2,70	0,014
26	27	28	200,00	12"	298,95	73,70	1,05	2,70	0,014
27	28	29	197,50	12“	298,95	59,20	0,84	1,81	0,015
28	29	30	200,00	12"	298,95	59,20	0,84	1,81	0,015
29	30	31	200,00	12"	298,95	42,50	0,61	1,00	0,016
30	31	32	200,00	12"	298,95	42,50	0,61	1,00	0,016
31	32	33	198,60	12“	298,95	42,50	0,61	1,00	0,016
34	3	21	100,00	6"	155,32	8,60	0,45	1,31	0,019
4	3	24	243,00	12"	298,95	85,00	1,21	3,50	0,014
35	7	22	152,40	16"	375,16	13,07	0,12	0,04	0,021
36	35	4	112,00	16“	375,16	108,90	0,99	1,84	0,014
37	21	23	100,00	6"	155,32	8,60	0,45	1,31	0,019
Bemba 2	2	3	•	•	•	202,50	0,00	30,7	0,000

2.6. Variables y restricciones

Las restricciones se describen como ecuaciones hidráulicas de conservación de la masa y la energía, pues toda red de distribución de agua está sujeta a las leyes inmodificables de la física.

Para agilizar los cálculos en el proceso de optimización, se garantiza el cumplimiento estricto de conservación de la masa y la energía conectando el algoritmo de

optimización diseñado con un paquete de simulación hidráulica de la red (EPANET) en el cual se cumplen intrínsecamente estas restricciones y cuyos resultados retroalimentan el proceso de optimización.

Adicionalmente, es necesario tomar como restricciones de diseño: la presión y la velocidad mínima requerida y máxima permitida en los nodos y las tuberías de la red; los valores límite de cada una de estas variables están contenidos en la resolución 0330 (Ministerio de Vivienda Ciudad y Territorio, 2017).

Por último, existen restricciones comerciales dadas por los diámetros de la tubería fabricada.

2.6.1. Conservación de la masa

Para cada nodo de consumo $n \in V$, la ley de conservación de la masa debe satisfacerse. Esta ley establece que el volumen de agua por unidad de tiempo que ingresa por el nodo n es igual al volumen de agua saliente, en el mismo nodo (Pereyra et al., 2017).

$$-\sum Q_{in} + \sum Q_{out} + DM = 0$$

donde Q_{in} es el caudal que ingresa al nodo n, Q_{out} es el caudal saliente y DM es el consumo en dicho nodo.

2.6.2. Conservación de la energía

Para cada malla cerrada, donde el flujo de agua circula por cada tubería, debe satisfacerse la ley de conservación de la energía. La ley establece que la sumatoria de cada una de las pérdidas de carga para una malla cerrada debe ser igual a cero. Siendo

M el número de mallas y T el número de tuberías perteneciente a dicha malla (Pereyra et al., 2017).

$$\sum_{j=1}^{T} \Delta H_j = 0 \quad \forall T \ \in M$$

2.6.3. Presión en los nodos

La presión mínima en la red de distribución debe ser de 15 m.c.a. en sistemas con poblaciones de diseño de más de 12.500 habitantes. Para sistemas nuevos u optimizaciones, la presión máxima debe ser de 50 m.c.a. (Ministerio de Vivienda Ciudad y Territorio, 2017)

Con el fin de acercarnos a los resultados de presión obtenidos del modelo que hace parte del diseño inicial, se ajusta el algoritmo generalizando los límites de presión entre 20 y 50 m.c.a.

2.6.4. Velocidad del flujo en las tuberías

La velocidad de circulación del agua deberá ser lo suficientemente alta para evitar estancamientos en la tubería, los cuales provocarían problemas de depósito en las paredes, así como otros de tipo sanitario. Del mismo modo, la velocidad máxima de circulación también estará limitada a un valor, por encima del cual, se considera que podrían presentarse problemas de erosión en la tubería, además de aumentar la posibilidad de que se produzca un golpe de ariete.

La velocidad mínima debe ser de 0.5 m/s, mientras que la velocidad máxima no deberá sobrepasar los límites de velocidad recomendados para el material del ducto a

emplear y/o los accesorios correspondientes. (Ministerio de Vivienda Ciudad y Territorio, 2017)

Pese a que son varios los factores que intervienen en el rango adecuado de velocidades, Mora (2012) menciona que es posible generalizar los límites adecuados entre 0.5 y 2.5 metros/segundo.

2.6.5. Diámetro de las tuberías

El diámetro mínimo en las redes de distribución no deberá ser inferior a 75 mm (3 pulgadas) para sectores urbanos. Se deben realizar los cálculos necesarios que permitan garantizar que, con el diámetro interno real de la tubería seleccionada, se cumplan las condiciones mínimas de servicio establecidas (Ministerio de Vivienda Ciudad y Territorio, 2017).

Los diámetros solo pueden tomar los valores discretos dados por la disponibilidad de diámetros de cada fabricante de tuberías. Para el caso de estudio los diámetros comerciales disponibles son once, a saber: 3, 4, 6, 8, 10, 12, 14, 16, 18, 20 y 24 pulgadas.

Teniendo en cuenta que, la red diseñada cuenta con 33 tramos de tubería y para buscar su solución óptima el mercado ofrece 11 diámetros diferentes de tuberías, el tamaño del espacio de búsqueda es 11^{33} (aproximadamente $2{,}34\ x\ 10^{34}$).

2.7. Función de costos

Para evaluar el costo de la red es necesario generar una ecuación que relacione el número de tuberías, la longitud de la tubería, que es un dato dado, el costo por metro lineal para dicha tubería, incluyendo su instalación, y los diámetros comerciales disponibles en el mercado.

Según Saldarriaga & Mendoza (2010) la función de costos que será objeto del proceso de optimización generalmente puede expresarse mediante una relación potencial del diámetro así:

$$C = \sum_{i=1}^{NT} k\, L_i\, D_i^x$$

Donde NT es el número de tuberías que conforman la red de distribución de Agua Potable, L_i y D_i son la longitud y el diámetro del i-ésimo tubo de la red, k y x son parámetros que se determina por regresión teniendo en cuenta los costos de determinado material en función del costo.

Basados en los precios que, para el suministro, instalación y construcción por metro lineal de redes de acueducto, utilizó como referencia la Empresa de Acueducto de Bogotá en el año 2017 y que se encuentran consignados en su Sistema de Avalúo de Infraestructura - SAI (Ver tabla 3), utilizando una regresión potencial se determinan los parámetros k y x que harán parte de la función costos empleada en este estudio (Ver figura 4).

Tabla 3. Precio de suministro e instalación de tubería PVC RDE 26 por metro lineal de acuerdo a su diámetro nominal en el año 2017.

Diámetro Nominal pulg.	**Diámetro Interior mm**	**Costo Unitario $**
3	82,04	14.958,93
4	105,52	22.856,14
6	155,32	44.857,00
8	202,17	73.682,07
10	252,07	112.390,22
12	298,95	156.788,86

Diámetro Nominal pulg.	Diámetro Interior mm	Costo Unitario $
14	328,26	251.968,81
16	375,16	299.919,08
18	422,04	461.651,45
20	468,94	519.605,07
24	562,72	697.034,52

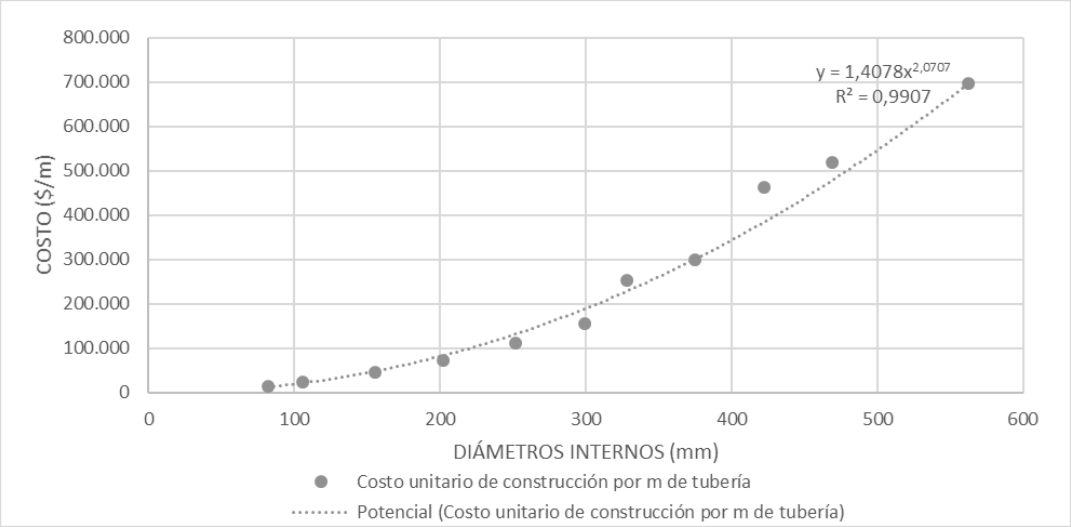


Figura 4. Curva función objetivo de costo para tubería de acuerdo con su diámetro comercial.

Fuente: Elaboración propia.

Donde:

$$k = 1{,}4078$$

$$x = 2{,}0707$$

$$C = \sum_{i=1}^{NT} 1{,}4078\, L_i\, D_i^{2{,}0707}$$

2.8. Materiales y métodos

2.8.1. EPANET

EPANET es un modelo de simulación por computador de distribución gratuita desarrollado por la División de Abastecimiento de Agua de la Agencia de Protección Ambiental de los Estados Unidos de Norteamérica (Environmental Protection Agency, EPA) que predice el comportamiento hidráulico del agua en redes de suministro a presión (Rossman, 2000) y que tiene gran aceptación en el mundo debido a que conjuga sus algoritmos de cálculo con una interfaz gráfica amigable y brinda la posibilidad de integrar su módulo de cálculo en otras aplicaciones.

Para el desarrollo de trabajos a medida, la EPA también ha desarrollado y proporcionado a los usuarios de EPANET una caja de herramientas (Toolkit de EPANET) que permite conectar la librería dinámica de cálculo, llamada API (Aplication Programming Interface por sus siglas en inglés), con algunos entornos de programación y resolver así ciertas problemáticas que en el software no se encuentran. La librería dinámica de EPANET v2.00.12 (API), está escrita en lenguaje ANSI C y posee módulos de código separados para el procesado de los datos de entrada, el análisis hidráulico, el análisis de calidad del agua, la resolución del sistema de ecuaciones lineales con matrices dispersas y la generación de informes (Vegas, Martínez, Alonso, & Tzatchkov, 2017).

Los lenguajes de programación más utilizados para trabajar con la Toolkit de EPANET son Basic 6.0, Basic .NET, Lenguaje de cálculo técnico (Matlab), C#, Python, C++.

2.8.2. Python

Python es un lenguaje de programación de propósito general, que se caracteriza por la simplicidad, versatilidad y rapidez de desarrollo. Es un lenguaje de scripting independiente de la plataforma (Windows, Mac, Linux, etc.) y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones en Windows a servidores de red o incluso, páginas web (Vegas et al., 2017).

La aplicación creada permite simulaciones de sistemas hidráulicos utilizando secuencias de comandos escritos en Python. Para este trabajo usamos las librerías EPANETTOOLS 1.0.0 y DEAP 1.3.0 escritas en lenguaje Python.

2.8.2.1. EPANETTOOLS

EPANETTOOLS 1.0.0 es un módulo preparado para usar las capacidades informáticas de EPANET en Python, para esto hace uso de la Toolkit de EPANET. Los modelos hidráulicos de las redes de suministro de agua en EPANET se proporcionan en archivos con extensión *.inp que contienen las características de tuberías, nodos, tanques de agua y componentes de control (bombas y válvulas). La aplicación utiliza la configuración de la red hidráulica almacenada en la base de datos, lo que también permite modificaciones.

La Toolkit de EPANET consta de 55 funciones y 104 variables que se utilizan con las funciones como parámetros de entrada. En la Tabla 4, se describen las funciones organizadas por tareas:

Tabla 4. Funciones de la Toolkit de Epanet agrupadas por tareas

Tarea	Funciones	
Ejecutar una simulación completa en una sola línea de comandos	ENepanet	
Abrir y cerrar la librería dinámica de Epanet	ENopen	Enclose
Recuperar información acerca de los nudos de la red	ENgetnodeindex ENgetnodeid	ENgetnodetype ENgetnodevalue

Tarea	Funciones	
Recuperar información acerca de las líneas de la red	ENgetlinkindex ENgetlinkvalue ENgetlinktype	ENgetlinknodes Engetlinkid
Recuperar información de las curvas de modulación	ENgetpatternid ENgetpatternindex	ENgetpatternlen ENgetpatternvalue
Obtener otra información de la red	ENgetcontrol ENgetqualtype ENgetoption ENgetversion	ENgetcount ENgetflowunits ENgettimeparam
Establecer nuevos valores para los parámetros de red	ENsetcontrol ENsetnodevalue ENsetlinkvalue ENaddpattern ENsetpattern	ENserpatternvalue ENsetqualtypc ENsettimeparam Ensetoption
Guardar y utilizar ficheros de resultados de análisis hidráulico	ENsavehydfile	Enusehyfile
Ejecutar un análisis hidráulico	ENsolveH ENopenH ENinitH	ENrunH ENncxtH EncloseH
Ejecutar un análisis de calidad del agua	ENsolveQ. ENopenQ. ENinitQ. ENrunQ.	ENnextQ. ENstepQ. ENcloseQ_
Generar un informe de salida	ENsaveH ENsaveinpfile ENreport ENresetreport	Ensetreport ENsetstatusreport Engeterror Enwriteline

2.8.2.2. DEAP (Distributed Evolutionary Algorithms in Python)

DEAP es un marco de cálculo evolutivo para la creación rápida de prototipos y pruebas de ideas. Su diseño, a diferencia de los marcos de caja negra más comunes, busca hacer que los algoritmos sean explícitos y que las estructuras de datos sean transparentes (Fortin, De Rainville, Gardner, Parizeau, & Gagné, 2012).

Su objetivo es proporcionar herramientas prácticas para la creación rápida de prototipos de algoritmos evolutivos personalizados, donde cada paso del proceso es explícito, fácil de leer y comprender.

El núcleo de DEAP se compone de dos estructuras simples: un creador y una caja de herramientas. El módulo creador es una fábrica que permite la creación de clases a través de la herencia o de la composición.

Los atributos, tanto datos como funciones, se pueden agregar dinámicamente a las clases existentes para crear nuevos tipos habilitados con funcionalidades de Computación Evolutiva especificadas por el usuario. Esto permite la creación de genotipos y poblaciones a partir de estructuras de datos como listas, conjuntos, diccionarios, árboles, etc. Este concepto de creador es clave para facilitar la implementación de cualquier tipo de algoritmo evolutivo, incluidos los algoritmos genéticos, la programación genética, las estrategias de evolución y la optimización de enjambre de partículas (Particle Swarm Optimization, PSO).

En DEAP la toolbox es un contenedor para las herramientas (operadores) que podrán usarse en un algoritmo evolutivo.

Las funcionalidades centrales de DEAP están apalancadas por varios módulos periféricos. El módulo de algoritmos contiene tres algoritmos de computación evolutiva clásicos: generacional, preguntar y contar (Collette, Hansen, Pujol, Salazar Aponte, & Le Riche, 2010); estos sirven como punto de partida para que la creación de algoritmos evolutivos personalizados que satisfagan necesidades específicas. El módulo de herramientas que proporciona operadores de computación evolutiva básicos como: inicializaciones, mutaciones, cruces y selecciones; estos operadores se pueden agregar directamente a una toolbox para ser utilizados posteriormente y contiene una serie de componentes que recopilan información útil sobre la evolución: estadísticas de estado físico, genealogía, salón de la fama para los mejores individuos encontrados hasta el momento y mecanismos de puntos de control para permitir el reinicio de la evolución. Un módulo base que contiene algunas estructuras de datos que se usan con frecuencia en computación evolutiva, pero que no se implementan en Python estándar (por ejemplo, objeto genérico de aptitud). El último módulo llamado DTM (Distributed Task Manager), ofrece sustitutos distribuidos para funciones comunes de Python como apply y map. DTM permite la distribución de partes específicas de los algoritmos al ocuparse de generar y distribuir subtareas. Incluso

equilibra la carga de trabajo entre los trabajadores para optimizar la eficiencia de distribución.

2.8.3. Función de evaluación y restricciones

Por practicidad se crea un solo código que contiene las características de la función de evaluación o costos y las restricciones descritas en el presente documento, en los numerales 2.7 y 2.6 respectivamente.

Inicialmente es necesario importar, total o parcialmente las librerías ***numpy*** (librería fundamental para el cálculo científico), ***os*** (permite acceder funcionalidades del Sistema Operativo) y ***EPANETTOOLS*** (interfaz en Python para EPANET).

```
#importar dependencias
import numpy as np
import os

from  epanettools.epanettools import EPANetSimulation, Node, Link, Network, Nodes,Links, Patterns, Controls, Control
from epanettools import epanet2 as epa
from os import remove
```

Posteriormente, se define la función *evalfit2* la cual contendrá tanto la función de costos como las penalizaciones por velocidad en las tuberías y presión en los nodos, los parámetros de ingreso a esta función provienen de los algoritmos de optimización y hacen referencia a los diámetros de la tubería después de cada ciclo en el algoritmo (*features*) y a la simulación completa del archivo original de la red en EPANET (*es*).

Adicionalmente, dentro de la función *evalfit2* se declaran las variables: *diámetro*, *longitudes*, *velocidades*, *presiones*, *cant_link* y *cant_node*, las cuales describen las características de los elementos que componen la red a analizar; también se declaran las listas: *VA*, que hace referencia al valor de cada tramo de tubería que conforma la red, *PEN1*, donde se ubican las penalizaciones por presiones en los nodos que se encuentren fuera de los límites establecidos, y *PEN2*, que almacena las

penalizaciones por velocidad fuera del rango de velocidades establecido para cada tramo de tubería.

```
def evalfit2(features,es):

    diametros=Link.value_type['EN_DIAMETER']
    longitudes=Link.value_type['EN_LENGTH']
    velocidades=Link.value_type['EN_VELOCITY']
    presiones=Node.value_type['EN_PRESSURE']

    cant_link=es.network.links #cantidad de tubos
    cant_node=es.network.nodes #cantidad de nodos

    VA=[]
    PEN1=[]
    PEN2=[]
```

Para evaluar el costo total de un individuo (red completa) se creó un ciclo que modifica el diámetro y aplica la función de costos descrita en el numeral 2.7 a cada uno de los tramos que conforman la red. El costo de cada tramo ingresa a la lista *VA* y se suma para obtener el valor total de la nueva red.

```
for id in range (len(cant_link)):
    r=(es.ENsetlinkvalue((id+1),diametros,features[id]))
    f=os.path.join('parcial_modificada.inp')
    a=((es.ENgetlinkvalue(id+1,longitudes)[1])*(1.4078*(es.ENgetlinkvalue(id+1,diametros)[1])**2.0707))
    VA.append(a)
cost = np.sum(VA)
```

Como se puede observar el ciclo guarda un archivo temporal denominado *parcial_modificada.inp*. Para ejecutar una simulación completa de la red contenida en el archivo temporal se utiliza la función *ENsolveH*.

```
es.ENsolveH();
```

El cumplimiento de las restricciones relacionadas con la conservación de la masa y de la energía se garantiza conectando los algoritmos de optimización propuestos con un paquete de simulación hidráulica de la red (EPANET) en el cual se cumplen intrínsecamente estas restricciones y cuyos resultados retroalimentan el proceso de optimización.

Para computar las penalizaciones relacionadas con la velocidad en las tuberías y la presión en los nodos, se construyen dos ciclos que permiten revisar los valores de estas variables.

```
for id in range (1,len(cant_node)):
    if (es.ENgetnodevalue(id,presiones)[1]) < 20:
        pen_pre = (20-(es.ENgetnodevalue(id,presiones)[1]))*100000000
    elif (es.ENgetnodevalue(id,presiones)[1]) > 50:
        pen_pre = ((es.ENgetnodevalue(id,presiones)[1])-50)*100000000
    else:
        pen_pre = 0
    PEN1.append(pen_pre)
PP=np.sum(PEN1)

for id in range (1,len(cant_link)):
    if (es.ENgetlinkvalue(id,velocidades)[1]) < 0.5:
        pen_vel = (0.5-(es.ENgetlinkvalue(id,velocidades)[1]))*100000000
    elif (es.ENgetlinkvalue(id,velocidades)[1]) > 2.5:
        pen_vel = ((es.ENgetlinkvalue(id,velocidades)[1])-2.5)*100000000
    else:
        pen_vel = 0
    PEN2.append(pen_vel)
PV=np.sum(PEN2)

costo=round(cost+PP+PV)

return costo,
remove('parcial_modificada.inp')
```

Cuando los valores de velocidad en las tuberías o presión en los nodos son inferiores o superiores a los límites inferior o superior establecidos para las mencionadas variables, la diferencia entre el límite y el valor obtenido de la simulación se multiplica por un valor lo suficientemente grande (10^8) que permita rechazar todas las soluciones que no cumplan las restricciones.

Finalmente, la función *evalfit2* retorna a los algoritmos de optimización el costo de la red más las penalizaciones por presión y por velocidad.

2.8.4. Algoritmo genético

Para optimizar el diseño de la red agua potable propuesta, en primera instancia se definen los individuos conformados por listas de diámetros escogidos al azar, luego se genera una población de individuos y por último se incluyen algunas funciones y operadores que se encargan de la evaluación y la evolución de la población.

Inicialmente, se importan parcial o totalmente algunas librerías y módulos, a saber: ***numpy*** (librería fundamental para el cálculo científico), ***os*** (permite acceder funcionalidades del Sistema Operativo), ***EPANETTOOLS*** (interfaz en Python para EPANET), ***pandas*** (permite realizar análisis de datos), ***random*** (contiene una serie de funciones relacionadas con la generación de valores aleatorios), ***matplotlib*** (permite crear gráficos), ***pyfiglet*** (crea textos con diferentes fuentes), ***time*** (permite medir tiempos) y ***deap*** (marco útil en el desarrollo de algoritmos evolutivos); además se importa el archivo que contiene la ecuación de evaluación de costos y penalizaciones *EC_EVAL*.

```
import pandas as pd
import numpy as np
import os
import random
import matplotlib.pyplot as plt
import pyfiglet

import EC_EVAL

from epanettools.epanettools import EPANetSimulation, Node, Link, Network, Nodes,Links, Patterns, Controls, Control
from time import time
from deap import creator, base, tools, algorithms
```

Dado que la estructura de los individuos requeridos en el algoritmo genético depende de la tarea que se pretende ejecutar, DEAP proporciona un método para crear contenedores de atributos asociados con el fitness (estado físico), llamado *DEAP.creator*. Usando este método es posible crear individuos personalizados de una manera muy simple (De Rainville, Fortin, Gardner, Parizeau, & Gagné, 2012).

El *creator* es una fábrica de clases que puede construir nuevas clases en tiempo real de ejecución. Primero se denomina a la nueva clase con el nombre deseado, luego la clase base heredará cualquier argumento posterior que desee convertir en atributos de su clase. Esto permite construir estructuras nuevas y complejas de cualquier tipo de contenedor, desde listas hasta árboles n-arios (De Rainville et al., 2012).

```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)
```

Inicialmente se define la clase: *FitnessMin*, la cual hereda del módulo *DEAP.base* la *Fitnessclase* y contendrá un atributo adicional llamado *weights*.

La primera línea crea un estado físico minimizador al reemplazar, en el tipo base *Fitness*, el atributo de pesos virtuales por *(-1.0,)* lo que implica minimizar un estado físico. La segunda línea crea una clase *Individual* que hereda las propiedades de la lista y tiene el atributo de aptitud del tipo *FitnessMin* que se acaba de crear.

En este último paso, es de resaltar cómo el *FitnessMin* recién creado se puede usar directamente como si fuera parte del creador.

Todos los objetos que se usan en el algoritmo propuesto: individuos, poblaciones y todas las funciones, operadores y argumentos, se almacenarán en un contenedor DEAP llamado *Toolbox*; este contiene dos métodos para agregar y eliminar contenido, *register() y unregister()* respectivamente.

```
DiametrosComerciales=(82.04, 105.52, 155.32, 202.17, 252.07, 298.95, 328.26, 375.16, 422.04, 468.94, 562.72)

toolbox = base.Toolbox()

toolbox.register("diam", random.choice, DiametrosComerciales)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.diam, n=DIM)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
```

En el anterior bloque de código se registra una función de generación y dos funciones de inicialización. Cuando se llama al generador *toolbox.diam()*, este escoge al azar un número de una lista dada (*DiametrosComerciales*). Los dos inicializadores por su parte producirán individuos y poblaciones.

El registro de herramientas en la *toolbox* solo asocia un alias a una función ya existente y congela parte de sus argumentos. Esto permite fijar una cantidad arbitraria de argumentos en ciertos valores, por lo que solo se especifican los restantes al llamar al método.

En el algoritmo propuesto los individuos se generan usando la función *initRepeat()*. Su primer argumento es la clase contenedor denominada *Individual* definida anteriormente. Este contenedor se llena utilizando el método *diam()*, proporcionado como segundo argumento, y contendrá la cantidad de tramos de tubería que hacen parte de la red a optimizar (variable *DIM*), como se especifica utilizando el tercer argumento.

Finalmente, el método *population()* usa el mismo modelo, pero difiere en que aún no se define el número de individuos que debería contener.

La función de evaluación para nuestro algoritmo fue descrita en el numeral anterior por lo que solo es necesario llamarla en cada iteración.

```
def evalfit(individual):
    return EC_EVAL.evalfit2(features=individual,es=es)
```

Dentro de DEAP hay dos formas de usar operadores. Puede simplemente llamarse como una función desde el módulo *tools* o registrarse con sus argumentos en una caja de herramientas. La forma más conveniente de hacerlo es registrar los operadores en la caja de herramientas, ya que esto permite cambiar fácilmente entre ellos.

El registro de los operadores y sus argumentos predeterminados en la caja de herramientas, se realizó de la siguiente manera:

```
toolbox.register("evaluate", evalfit)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)
```

La evaluación recibe el alias *evaluate*. Al tener un único argumento que es el individuo para evaluar no necesita ningún arreglo, el individuo se da más adelante en el algoritmo. La mutación, por su parte, necesita como argumento la probabilidad

independiente de que cada atributo sea mutado *indpb*. En el algoritmo, la función *mutate()* se llama como sigue: *toolbox.mutate*. Esta es la forma más conveniente porque permite que cada mutación tome un número diferente de argumentos.

Después de elegir la representación y los operadores, se escribe propiamente el algoritmo dentro de una función llamada *main()*.

Para iniciar la población se utiliza el método registrado anteriormente en la caja de herramientas.

```
def main():

    poblacion = int(input("Ingrese cantidad de individuos - poblacion inicial: "))
    pop = toolbox.population(n=poblacion)
```

Donde *pop* será una *list* compuesta por una cantidad de individuos que define el usuario del algoritmo al correrlo; esto dado que el parámetro *n* quedó abierto durante el registro del método *population()* en la caja de herramientas.

Para verificar el rendimiento de la evolución, se registran, calculan y guardan los valores mínimos, máximos y medios de las condiciones físicas de todos los individuos de la población, así como sus desviaciones estándar.

```
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("media", np.mean)
stats.register("desv est", np.std)
stats.register("min", np.min)
stats.register("max", np.max)

logbook = tools.Logbook()
logbook.header = ["gen", "evals"] + stats.fields
```

A continuación, se evalúa la nueva población.

```
fitnesses = list(map(toolbox.evaluate, pop))
for ind, fit in zip(pop, fitnesses):
    ind.fitness.values = fit
```

Primero se asigna la función *map()* para evaluar cada individuo, luego se asigna el estado físico respectivo.

Posteriormente, el código creado permite que el usuario defina las probabilidades de cruce y de mutación (constantes *CXPB* y *MUTPB*) que serán utilizadas más adelante.

```
CXPB = float(input("Ingrese la probabilidad de cruce: "))
MUTPB = float(input("Ingrese la probabilidad de mutacion: "))
```

Recordando que los individuos creados consisten en 34 números seleccionados de la lista de diámetros comerciales y que la finalidad del algoritmo es evolucionar la población hasta encontrar al individuo que representa un menor costo constructivo de la red, es claro que solo pretenden obtenerse los valores de aptitud (costo) de cada individuo.

A continuación, se crean las variables necesarias para llevar el conteo del número de generaciones y establecer el criterio de parada para el algoritmo ($while\ n\ <\ 100$).

```
# Seguimiento variable del número de generaciones
g = 0

# Variable - Criterio de parada
n=0

MIN=[]

tiempo_inicial = time()

# Comienza la evolución
while n < 100:

    # Una nueva generación
    g = g + 1
```

La evolución en sí misma se realiza seleccionando, apareando y mutando a los individuos que hacen parte de la población.

En el algoritmo genético creado, el primer paso es seleccionar la próxima generación.

```
# Seleccione los individuos de la siguiente generación
offspring = toolbox.select(pop, len(pop))
# Clonar los individuos seleccionados
offspring = list(map(toolbox.clone, offspring))
```

El anterior fragmento de código creará una lista de descendientes, que es una copia exacta de los individuos seleccionados. El método *toolbox.clone()* asegura que no se usen como referencia los individuos sino una instancia completamente independiente.

A continuación, se realiza tanto el crossover (apareamiento) como la mutación de los descendientes producidos con una cierta probabilidad de cruce (*CXPB*) y mutación (*MUTPB*). La declaración *del* invalidará la aptitud de la descendencia modificada.

```
# Aplicar cruce y mutación en la descendencia
for child1, child2 in zip(offspring[::2], offspring[1::2]):

    # cruzar dos individuos con probabilidad CXPB
    if random.random() < CXPB:
        toolbox.mate(child1, child2)

        # valores de condición física de los niños, debe recalcularse más tarde
        del child1.fitness.values
        del child2.fitness.values

for mutant in offspring:

    # mutar un individuo con probabilidad MUTPB
    if random.random() < MUTPB:
        toolbox.mutate(mutant)
        del mutant.fitness.values
```

Los operadores de cruce (o apareamiento) y de mutación proporcionados por DEAP, generalmente toman como entrada 1 o 2 individuos y devuelven 1 o 2 individuos modificados; al modificar los individuos dentro del contenedor de la caja de herramientas, se hace innecesario reasignar sus resultados.

Dado que el contenido de algunos de los descendientes cambió durante el último paso, se reevalúa su estado físico. Para ahorrar tiempo y recursos, solo se mapea a aquellos descendientes cuyos ejercicios se marcaron como no válidos.

```
# Evaluar a los individuos con una condición física inválida
invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
fitnesses = map(toolbox.evaluate, invalid_ind)
for ind, fit in zip(invalid_ind, fitnesses):
    ind.fitness.values = fit
```

Por último, se reemplaza la antigua población por la descendencia.

```
# La población es reemplazada enteramente por la descendencia.
pop[:] = offspring
```

Para finalizar se imprimen las estadísticas de cada evolución, se aplica el criterio de parada y se calcula el tiempo de ejecución del algoritmo.

```
    logbook.record(gen=g, evals=len(invalid_ind), **stats.compile(pop))
    print(logbook.stream)
    MIN.append(min(fits))

    for i in range (g-1,g):
        if MIN [i-1]==MIN[i]:
            n=n+1
        else:
            n=0

tiempo_final = time()

tiempo_ejecucion = round(tiempo_final - tiempo_inicial, ndigits=2)
```

2.8.5. PSO (Particle Swarm Optimization)

El algoritmo PSO optimiza un problema a parir de una población de soluciones candidatas denominadas partículas, las cuales mueve a través de un espacio de búsqueda utilizando fórmulas matemáticas simples. Los movimientos de las partículas son guiados por las mejores posiciones encontradas en el espacio de búsqueda; las posiciones se actualizan teniendo en cuenta las mejores posiciones encontradas por las partículas.

Antes de escribir funciones y algoritmos, es necesario importar, además de las librerías descritas con anterioridad, el modúlo ***operator*** (permite acceder a los operadores matemáticos, lógicos, relacionales, bit a bit, etc. intrínsecos de Python).

```
#importar dependencias
import pandas as pd
import numpy as np
import os
import random
import operator
import pyfiglet
import matplotlib.pyplot as plt
import math

import EC_EVAL

from  epanettools.epanettools import EPANetSimulation, Node, Link, Network, Nodes,Links, Patterns, Controls, Control
from time import time
from deap import creator, base, tools, algorithms
```

El objetivo de la partícula es minimizar el valor de retorno de la función en su posición.

Las partículas de PSO se describen esencialmente como posiciones en un espacio de búsqueda de dimensiones *D*. Cada partícula también tiene un vector que representa la velocidad de la partícula en cada dimensión. Finalmente, cada partícula mantiene una referencia al mejor estado en el que ha estado hasta ahora.

Esto se traduce en DEAP en las siguientes dos líneas de código:

```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Particle", list, fitness=creator.FitnessMin, speed=list,
    smin=None, smax=None, best=None)
```

El anterior fragmento de código crea dos nuevos objetos en el espacio *creator*. Primero, crea un objeto *FitnessMin* y se especifica el *weights (-1.0,)*, lo que significa que el objetivo del algoritmo es minimizar el valor de la aptitud de las partículas. El segundo objeto que se crea representa a la partícula. La cual se define como una lista que contiene cinco atributos. El primer atributo es la aptitud de la partícula, el segundo es la velocidad de la partícula que también será una lista, el tercero y el

cuarto son el límite del valor de la velocidad, y el quinto atributo es una referencia a una copia del mejor estado de la partícula hasta ahora. Como la partícula no tiene un estado final hasta que se evalúe, la referencia se establece en *None*. Los límites de velocidad también están referenciados *None* para permitir la configuración a través de la función *generate()* que se describe más adelante.

El algoritmo PSO utiliza tres operadores: inicializador, actualizador y evaluador. La inicialización consiste en generar una posición y una velocidad aleatoria para una partícula. La siguiente función crea una partícula e inicializa sus atributos, excepto el atributo *best*, que se establecerá solo después de la evaluación:

```
def generate(size, smin, smax):
    part = creator.Particle(random.choice(DiametrosComerciales) for _ in range(size))
    part.speed = [random.uniform(smin, smax) for _ in range(size)]
    part.smin = smin
    part.smax = smax
    return part
```

La función *updateParticle()* primero calcula la velocidad, luego limita los valores de velocidad entre *smin* y *smax*, y finalmente calcula la nueva posición de la partícula.

```
def updateParticle(part, best, phi1, phi2):
    u1 = (random.uniform(0, phi1) for _ in range(len(part)))
    u2 = (random.uniform(0, phi2) for _ in range(len(part)))
    v_u1 = map(operator.mul, u1, map(operator.sub, part.best, part))
    v_u2 = map(operator.mul, u2, map(operator.sub, best, part))
    part.speed = list(map(operator.add, part.speed, map(operator.add, v_u1, v_u2)))
    for i, speed in enumerate(part.speed):
        if abs(speed) < part.smin:
            part.speed[i] = math.copysign(part.smin, speed)
        elif abs(speed) > part.smax:
            part.speed[i] = math.copysign(part.smax, speed)
    part[:] = list(map(suma, part, part.speed))
    return part
```

Los operadores están registrados en la caja de herramientas con sus parámetros. La velocidad está limitada en el rango *[-50, 50]* a través de toda la evolución.

La función de evaluación *evalfit(Particle)* ya está definida en el módulo *EC_EVAL*, por lo que podemos registrarla directamente.

```
def evalfit(Particle):
    return EC_EVAL.evalfit2(features=Particle,es=es)
```

Con el fin de discretizar el algoritmo, las funciones *closest* y *suma* toman las nuevas posiciones adquiridas por cada componente de las partículas y las acerca al diámetro comercial más cercano.

```
def closest(K):
    lst=(82.04, 105.52, 155.32, 202.17, 252.07, 298.95, 328.26, 375.16, 422.04, 468.94, 562.72)
    r=lst[min(range(len(lst)), key = lambda i: abs(lst[i]-K))]
    return r


def suma(i, j):
    resultado=i+j
    resultado=closest(resultado)
    return resultado
```

A continuación, se registran los operadores en la caja de herramientas.

```
toolbox = base.Toolbox()
toolbox.register("Particle", generate, size=DIM, smin=-50, smax=50)
toolbox.register("population", tools.initRepeat, list, toolbox.Particle)
toolbox.register("update", updateParticle, phil=2.0, phi2=2.0)
toolbox.register("evaluate", evalfit)
```

Para activar el algoritmo primero se crea una población y luego se aplica el algoritmo *PSO*. La variable *best* contiene la mejor partícula encontrada.

```python
def main():

    poblacion = int(input("Ingrese cantidad de particulas - poblacion inicial: "))

    pop = toolbox.population(n=poblacion)

    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", np.mean)
    stats.register("std", np.std)
    stats.register("min", np.min)
    stats.register("max", np.max)

    logbook = tools.Logbook()
    logbook.header = ["gen", "evals"] + stats.fields

    best = None
    g = 0    # Seguimiento variable del número de generaciones
    n=0  # Variable - Criterio de parada
    MIN=[]  # Lista de minimos

    tiempo_inicial = time()

    while n < 100:

        g = g + 1

        for part in pop:
            part.fitness.values = toolbox.evaluate(part)
            if not part.best or part.best.fitness < part.fitness:
                part.best = creator.Particle(part)
                part.best.fitness.values = part.fitness.values
            if not best or best.fitness < part.fitness:
                best = creator.Particle(part)
                best.fitness.values = part.fitness.values
        for part in pop:
            toolbox.update(part, best)

        fits = [ind.fitness.values[0] for ind in pop]

        # Reuna todos los ejercicios en una lista e imprima las estadísticas
        logbook.record(gen=g, evals=len(pop), **stats.compile(pop))
        print(logbook.stream)

        MIN.append(min(fits))

        # Criterio de parada
        for i in range (g-1,g):
            if MIN [i-1]==MIN[i]:
                n=n+1
            else:
                n=0

    tiempo_final = time()

    tiempo_ejecucion = round(tiempo_final - tiempo_inicial, ndigits=2)
```

Capítulo III

Resultados y contribución

3.1. Costo de la red antes de optimizar

Para poder analizar la conveniencia de aplicar o no métodos de optimización en el diseño original de la red de acueducto construida en el municipio de Mosquera, se aplicó la ecuación de costos utilizada en los algoritmos propuestos a cada tramo de tubería, utilizando el diámetro realmente utilizado durante la construcción del sistema (Ver tabla 5).

Tabla 5. Características y costo constructivo de la red de acueducto antes de optimizar

Tramo	Nodos		Long.	Diámetro		Costo
				Nominal	Interno	
	De	A	M	pulg	Mm	$
1	1	2	322	18"	422,04	123.798.836
3	3	35	85,2	20"	468,94	40.743.948
6	4	5	535,5	16“	375,16	161.335.745
7	5	7	312,8	16"	375,16	94.240.562
8	6	8	597,3	16"	375,16	179.954.884
9	6	8	150,6	6"	155,32	7.307.023
10	8	9	100	6"	155,32	4.851.941
11	8	10	300	6"	155,32	14.555.823
12	11	10	24,7	6"	155,32	1.198.429
13	11	12	317,4	6"	155,32	15.400.060
14	22	11	347,8	6"	155,32	16.875.050
15	5	14	1156	12"	298,95	217.630.985
16	14	15	554,6	10"	252,07	73.341.653
17	15	16	277,8	10"	252,07	36.736.948
18	16	17	102,9	6"	155,32	4.992.647
19	17	18	332,9	6"	155,32	16.152.111
20	18	19	100	4"	105,52	2.179.015
21	16	20	100	8"	202,17	8.375.072
22	4	13	239,4	8"	202,17	20.049.923
23	24	25	300	12“	298,95	56.478.629

Tramo	Nodos		Long.	Diámetro		Costo
				Nominal	Interno	
	De	A	M	pulg	Mm	$
24	25	26	300	12"	298,95	56.478.629
25	26	27	362,5	12"	298,95	68.245.011
26	27	28	200	12"	298,95	37.652.420
27	28	29	197,5	12“	298,95	37.181.764
28	29	30	200	12"	298,95	37.652.420
29	30	31	200	12"	298,95	37.652.420
30	31	32	200	12"	298,95	37.652.420
31	32	33	198,6	12“	298,95	37.388.853
34	3	21	100	6"	155,32	4.851.941
4	3	24	243	12"	298,95	45.747.690
35	7	22	152,4	16"	375,16	45.915.159
36	35	4	112	16“	375,16	33.743.424
37	21	23	100	6"	155,32	4.851.941
Costo total de la red ($)						1.581.213.376

3.2. Algoritmo Genético

Durante la aplicación del Algoritmo Genético desarrollado, se encontraron diferentes alternativas de solución al diseño de la red propuesta, las cuales presentan en todos los casos un costo inferior al de la red originalmente diseñada y mejoran sus características hidráulicas cumpliendo casi en su totalidad con las restricciones de velocidad en las tuberías y presión en los nodos. La Tabla 6 resume los costos obtenidos, la cantidad de iteraciones y el tiempo requerido para encontrar esa solución.

Tabla 6. Soluciones obtenidas aplicando Algoritmo Genético

Solución	Población inicial	Costo total de la red optimizada	Cantidad de iteraciones	Tiempo de ejecución	Porcentaje de reducción del costo
Id	Individuos	$	Generaciones	Seg	%
1	100	1.106.872.475	322	92,64	30,00
2	100	1.059.335.907	288	86,03	33,00

Solución	Población inicial	Costo total de la red optimizada	Cantidad de iteraciones	Tiempo de ejecución	Porcentaje de reducción del costo
Id	**Individuos**	**$**	**Generaciones**	**Seg**	**%**
3	100	1.057.540.960	348	101,83	33,12
4	100	1.056.434.222	311	90,79	33,19
5	100	1.048.099.944	331	106,11	33,72
6	100	1.030.672.995	314	87,57	34,82
7	100	1.187.222.603	334	93,78	24,92
8	100	1.057.417.463	408	124,7	33,13
9	100	1.046.980.160	308	87,67	33,79
10	100	1.022.784.630	249	72,53	35,32
Promedio		1.067.336.136	321,3	94,37	32,50

3.2.1. Solución de costo mínimo con AG

Aunque todas las soluciones obtenidas al aplicar el Algoritmo Genético proporcionan un costo inferior al del diseño original elaborado por Integrita S.A.S. (2014), la solución cuyo valor es mínimo ocurre en la décima ejecución del Algoritmo Genético, esta se obtiene tras 72,53 segundos de ejecución y 249 generaciones, y reduce el costo total de construcción de la red de acueducto en un 35,32%. A continuación, en las tablas 7, 8 y 9 se resumen los costos de cada tramo de la red modificada y sus principales características hidráulicas; adicionalmente en la figura 4 se ven representadas la velocidad en las tuberías y la presión en los nodos que conforman la red.

Tabla 7. Características y costo constructivo de la red de acueducto optimizada con AG

#	Nodos		Long.	Diámetro		Costo
				Nominal	Interno	
	De	**A**	**M**	**pulg**	**Mm**	**$**
1	1	2	322	18	422,04	123.798.836
3	3	35	85,2	14	328,26	19.467.631
6	4	5	535,5	12	298,95	100.814.354

#	Nodos		Long.	Diámetro		Costo
				Nominal	Interno	
	De	A	M	pulg	Mm	$
7	5	6	312,8	8	202,17	26.197.226
8	6	7	597,3	4	105,52	13.015.259
9	6	8	150,6	6	155,32	7.307.023
10	8	9	100	3	82,04	1.293.939
11	8	10	300	4	105,52	6.537.046
12	11	10	24,7	6	155,32	1.198.429
13	11	12	317,4	4	105,52	6.916.195
14	22	11	347,8	4	105,52	7.578.615
15	5	14	1156	12	298,95	217.630.985
16	14	15	554,6	10	252,07	73.341.653
17	15	16	277,8	12	298,95	52.299.211
18	16	17	102,9	8	202,17	8.617.949
19	17	18	332,9	6	155,32	16.152.111
20	18	19	100	4	105,52	2.179.015
21	16	20	100	10	252,07	13.224.243
22	4	13	239,4	4	105,52	5.216.563
23	24	25	300	10	252,07	39.672.730
24	25	26	300	10	252,07	39.672.730
25	26	27	362,5	10	252,07	47.937.882
26	27	28	200	10	252,07	26.448.487
27	28	29	197,5	10	252,07	26.117.880
28	29	30	200	8	202,17	16.750.144
29	30	31	200	8	202,17	16.750.144
30	31	32	200	8	202,17	16.750.144
31	32	33	198,6	10	252,07	26.263.347
34	3	21	100	3	82,04	1.293.939
4	3	24	243	10	252,07	32.134.911
35	7	22	152,4	4	105,52	3.320.819
36	35	4	112	14	328,26	25.591.251
37	21	23	100	3	82,04	1.293.939
Costo Total de la Red						1.022.784.630

Tabla 8. Resultados de la modelación de la red de acueducto optimizada con AG (tuberías)

#	Nodos		Long.	Diámetro		Caudal	Velocidad	Pérd. Unit.	
				Nominal	Interno				
	De	A	m	pulg	Mm	LPS	m/s	m/km	
1	1	2	322	18	422,04	55,81	0,4	0,31	
3	3	35	85,2	14	328,26	108,9	1,29	3,5	
6	4	5	535,5	12	298,95	95,5	1,36	4,33	
7	5	6	312,8	8	202,17	29,2	0,91	3,32	
8	6	7	597,3	4	105,52	7,44	0,85	6,45	
9	6	8	150,6	6	155,32	21,76	1,15	6,93	
10	8	9	100	3	82,04	3,9	0,74	6,78	
11	8	10	300	4	105,52	13,96	1,6	19,98	
12	11	10	24,7	6	155,32	-10,36	0,55	1,82	
13	11	12	317,4	4	105,52	9,3	1,06	9,62	
14	22	11	347,8	4	105,52	7,44	0,85	6,45	
15	5	14	1156	12	298,95	66,3	0,94	2,23	
16	14	15	554,6	10	252,07	66,3	1,33	5,07	
17	15	16	277,8	12	298,95	63,2	0,9	2,04	
18	16	17	102,9	8	202,17	19	0,59	1,53	
19	17	18	332,9	6	155,32	12	0,63	2,38	
20	18	19	100	4	105,52	6	0,69	4,38	
21	16	20	100	10	252,07	44,2	0,89	2,43	
22	4	13	239,4	4	105,52	13,4	1,53	18,57	
23	24	25	300	10	252,07	73,7	1,48	6,15	
24	25	26	300	10	252,07	73,7	1,48	6,15	
25	26	27	362,5	10	252,07	73,7	1,48	6,15	
26	27	28	200	10	252,07	73,7	1,48	6,15	
27	28	29	197,5	10	252,07	59,2	1,19	4,13	
28	29	30	200	8	202,17	59,2	1,84	11,96	
29	30	31	200	8	202,17	42,5	1,32	6,55	
30	31	32	200	8	202,17	42,5	1,32	6,55	
31	32	33	198,6	10	252,07	42,5	0,85	2,26	
34	3	21	100	3	82,04	8,6	1,63	28	
4	3	24	243	10	252,07	85	1,7	7,97	
35	7	22	152,4	4	105,52	7,44	0,85	6,45	
36	35	4	112	14	328,26	108,9	1,29	3,5	
37	21	23	100	3	82,04	8,6	1,63	28	
2	2	3	NA	NA	NA	202,5	0	-30,7	Pump

Tabla 9. Resultados de la modelación de la red de acueducto optimizada con AG (nodos)

ID Nodo	Demanda	Altura	Presión	
	LPS	M	mca	
3	0	79,7	34,7	
4	0	79,01	34,01	
5	0	76,7	32,2	
6	0	75,66	31,63	
7	0	71,8	28,65	
8	3,9	74,61	30,61	
9	3,9	73,93	29,93	
10	3,6	68,62	24,82	
11	8,5	68,57	24,77	
12	9,3	65,52	21,32	
13	13,4	74,57	29,57	
14	0	74,12	28,22	
15	3,1	71,31	24,31	
16	0	70,74	22,74	
17	7	70,58	22,58	
18	6	69,79	21,59	
19	6	69,35	21,05	
20	44,2	70,5	22,3	
21	0	76,9	31,9	
23	8,6	74,1	29,1	
24	11,3	77,76	32,76	
25	0	75,92	31,12	
26	0	74,08	29,48	
27	0	71,85	27,95	
28	14,5	70,62	27,42	
29	0	69,8	27,3	
30	16,7	67,41	25,61	
31	0	66,1	24,4	
32	0	64,79	23,09	
33	42,5	64,34	22,74	
22	0	70,82	27,42	
35	0	79,4	34,4	
1	-55,81	49,1	0	Reservoir
2	-146,69	49	4	Tank

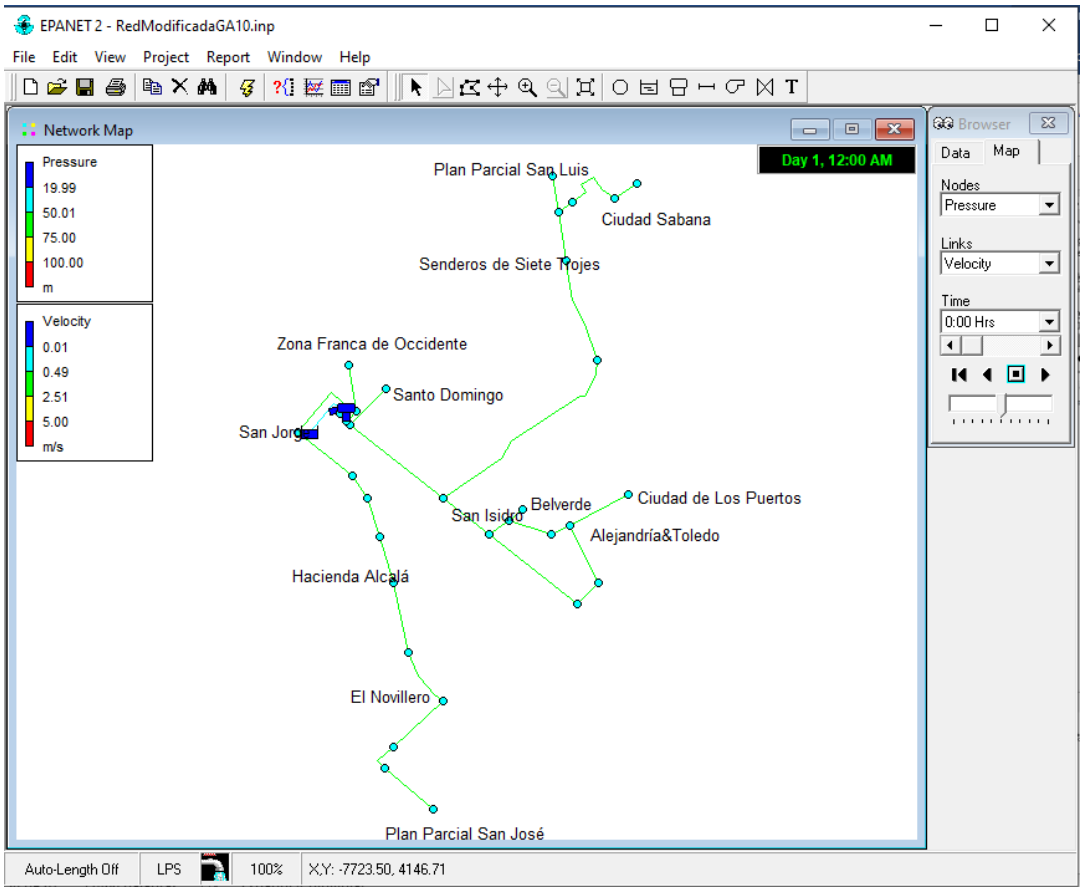


Figura 5. Esquema de la red optimizada utilizando AG – Velocidad en las tuberías y presiones en los nodos.

Fuente: Epanet.

3.3. PSO (Particle Swarm Optimization)

Al igual que con las alternativas de solución dadas tras la aplicación del Algoritmo Genético, el algoritmo PSO también encontró diferentes soluciones para el diseño de la red propuesta; estas, además de disminuir el costo de la red originalmente diseñada también mejoran en todos los casos sus características hidráulicas. La tabla 10 resume los costos obtenidos, la cantidad y el número de iteraciones necesario para encontrar esa solución.

Tabla 10. Soluciones obtenidas aplicando Particle Swarm Optimization

Solución	Población inicial	Costo total de la red optimizada	Cantidad de iteraciones	Tiempo de ejecución	Porcentaje de reducción del costo
Id	**Individuos**	**$**	**Iteraciones**	**seg**	**%**
1	100	996.556.325	470	217,8	36,98
2	100	1.046.048.315	535	247,73	33,85
3	100	994.929.505	480	221,66	37,08
4	100	1.031.302.977	517	243,67	34,78
5	100	1.013.721.257	546	249,18	35,89
6	100	1.005.108.303	406	156,6	36,43
7	100	998.876.851	492	175,52	36,83
8	100	1.003.075.745	663	300,88	36,56
9	100	1.017.173.275	541	237,76	35,67
10	100	1.013.930.602	584	269,98	35,88
Promedio		1.012.072.316	523,4	232,08	35,99

3.3.1. Solución de costo mínimo con PSO

A pesar que todas las soluciones obtenidas al aplicar el algoritmo PSO proporcionan un costo inferior al del diseño original, la solución cuyo valor es mínimo ocurre en la tercera ejecución del algoritmo PSO, esta se obtiene tras 221,66 segundos de ejecución y 480 generaciones, y reduce el costo total de construcción de la red de acueducto en un 37,08%. A continuación, en las tablas 11, 12 y 13 se resumen los costos de cada tramo de la red modificada y sus principales características hidráulicas; adicionalmente en la figura 6 se ven representadas la velocidad en las tuberías y la presión en los nodos que conforman la red.

Tabla 11. Características y costo constructivo de la red de acueducto optimizada con PSO

#	Nodos		Long.	Diámetro		Costo
				Nominal	Interno	
	De	A	M	Pulg	Mm	$
1	1	2	322	18	422,04	123.798.836
3	3	35	85,2	14	328,26	19.467.631
6	4	5	535,5	12	298,95	100.814.354
7	5	6	312,8	6	155,32	15.176.871
8	6	7	597,3	3	82,04	7.728.699
9	6	8	150,6	6	155,32	7.307.023
10	8	9	100	3	82,04	1.293.939
11	8	10	300	6	155,32	14.555.823
12	11	10	24,7	4	105,52	538.217
13	11	12	317,4	4	105,52	6.916.195
14	22	11	347,8	3	82,04	4.500.321
15	5	14	1156	12	298,95	217.630.985
16	14	15	554,6	10	252,07	73.341.653
17	15	16	277,8	12	298,95	52.299.211
18	16	17	102,9	8	202,17	8.617.949
19	17	18	332,9	6	155,32	16.152.111
20	18	19	100	4	105,52	2.179.015
21	16	20	100	8	202,17	8.375.072
22	4	13	239,4	4	105,52	5.216.563
23	24	25	300	10	252,07	39.672.730
24	25	26	300	10	252,07	39.672.730
25	26	27	362,5	10	252,07	47.937.882
26	27	28	200	10	252,07	26.448.487
27	28	29	197,5	10	252,07	26.117.880
28	29	30	200	8	202,17	16.750.144
29	30	31	200	8	202,17	16.750.144
30	31	32	200	8	202,17	16.750.144
31	32	33	198,6	8	202,17	16.632.893
34	3	21	100	3	82,04	1.293.939
4	3	24	243	10	252,07	32.134.911
35	7	22	152,4	3	82,04	1.971.963
36	35	4	112	14	328,26	25.591.251
37	21	23	100	3	82,04	1.293.939
Costo Total de la Red						994.929.505

Tabla 12. Resultados de la modelación de la red de acueducto optimizada con PSO (tuberías)

#	Nodos		Long.	Diámetro		Caudal	Velocidad	Pérd. Unit.	
				Nominal	Interno				
	De	A	m	pulg	Mm	LPS	m/s	m/km	
1	1	2	322	18	422,04	55,81	0,4	0,31	
3	3	35	85,2	14	328,26	108,9	1,29	3,5	
6	4	5	535,5	12	298,95	95,5	1,36	4,33	
7	5	6	312,8	6	155,32	29,2	1,54	11,81	
8	6	7	597,3	3	82,04	2,62	0,5	3,34	
9	6	8	150,6	6	155,32	26,58	1,4	9,96	
10	8	9	100	3	82,04	3,9	0,74	6,78	
11	8	10	300	6	155,32	18,78	0,99	5,32	
12	11	10	24,7	4	105,52	-15,18	1,74	23,27	
13	11	12	317,4	4	105,52	9,3	1,06	9,62	
14	22	11	347,8	3	82,04	2,62	0,5	3,34	
15	5	14	1156	12	298,95	66,3	0,94	2,23	
16	14	15	554,6	10	252,07	66,3	1,33	5,07	
17	15	16	277,8	12	298,95	63,2	0,9	2,04	
18	16	17	102,9	8	202,17	19	0,59	1,53	
19	17	18	332,9	6	155,32	12	0,63	2,38	
20	18	19	100	4	105,52	6	0,69	4,38	
21	16	20	100	8	202,17	44,2	1,38	7,03	
22	4	13	239,4	4	105,52	13,4	1,53	18,57	
23	24	25	300	10	252,07	73,7	1,48	6,15	
24	25	26	300	10	252,07	73,7	1,48	6,15	
25	26	27	362,5	10	252,07	73,7	1,48	6,15	
26	27	28	200	10	252,07	73,7	1,48	6,15	
27	28	29	197,5	10	252,07	59,2	1,19	4,13	
28	29	30	200	8	202,17	59,2	1,84	11,96	
29	30	31	200	8	202,17	42,5	1,32	6,55	
30	31	32	200	8	202,17	42,5	1,32	6,55	
31	32	33	198,6	8	202,17	42,5	1,32	6,55	
34	3	21	100	3	82,04	8,6	1,63	28	
4	3	24	243	10	252,07	85	1,7	7,97	
35	7	22	152,4	3	82,04	2,62	0,5	3,34	
36	35	4	112	14	328,26	108,9	1,29	3,5	
37	21	23	100	3	82,04	8,6	1,63	28	
2	2	3	NA	NA	NA	202,5	0	-30,7	Pump

Tabla 13. Resultados de la modelación de la red de acueducto optimizada con PSO (nodos)

ID Nodo	Demanda	Altura	Presión	
	LPS	m	mca	
3	0	79,7	34,7	
4	0	79,01	34,01	
5	0	76,7	32,2	
6	0	73	28,97	
7	0	71	27,85	
8	3,9	71,5	27,5	
9	3,9	70,82	26,82	
10	3,6	69,9	26,1	
11	8,5	69,33	25,53	
12	9,3	66,28	22,08	
13	13,4	74,57	29,57	
14	0	74,12	28,22	
15	3,1	71,31	24,31	
16	0	70,74	22,74	
17	7	70,58	22,58	
18	6	69,79	21,59	
19	6	69,35	21,05	
20	44,2	70,04	21,84	
21	0	76,9	31,9	
23	8,6	74,1	29,1	
24	11,3	77,76	32,76	
25	0	75,92	31,12	
26	0	74,08	29,48	
27	0	71,85	27,95	
28	14,5	70,62	27,42	
29	0	69,8	27,3	
30	16,7	67,41	25,61	
31	0	66,1	24,4	
32	0	64,79	23,09	
33	42,5	63,49	21,89	
22	0	70,49	27,09	
35	0	79,4	34,4	
1	-55,81	49,1	0	Reservoir
2	-146,69	49	4	Tank

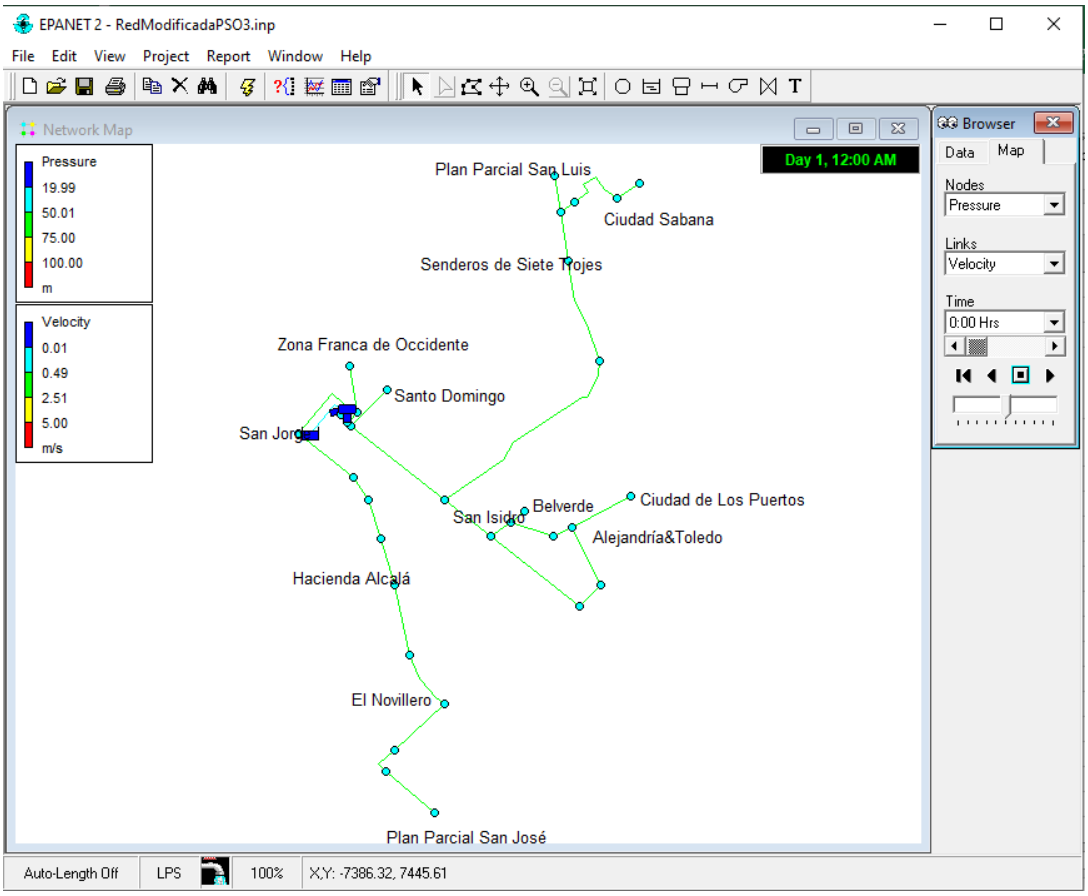


Figura 6. Esquema de la red optimizada utilizando PSO – Velocidad en las tuberías y presiones en los nodos.

Fuente: Epanet.

3.4. Discusión

La comparación de los resultados obtenidos al aplicar tanto el Algoritmo Genético diseñado como el algoritmo PSO en la red propuesta nos permite en primera instancia observar que la aplicación de cualquiera de los dos algoritmos reduce considerablemente los costos de la red inicialmente diseñada; en promedio el algoritmo PSO reduce el 35,99% y el Algoritmo Genético el 32,50%.

Los costos promedio producto de las optimizaciones ejecutadas con PSO son 5,18% más bajos que los obtenidos cuando se ejecuta la optimización con Algoritmos Genéticos.

A continuación, en la figura 7 se puede observar el costo de la red de acueducto antes de surtirse el proceso de optimización y los costos de cada una de las diez optimizaciones ejecutadas utilizando el Algoritmo Genético diseñado y el algoritmo PSO.

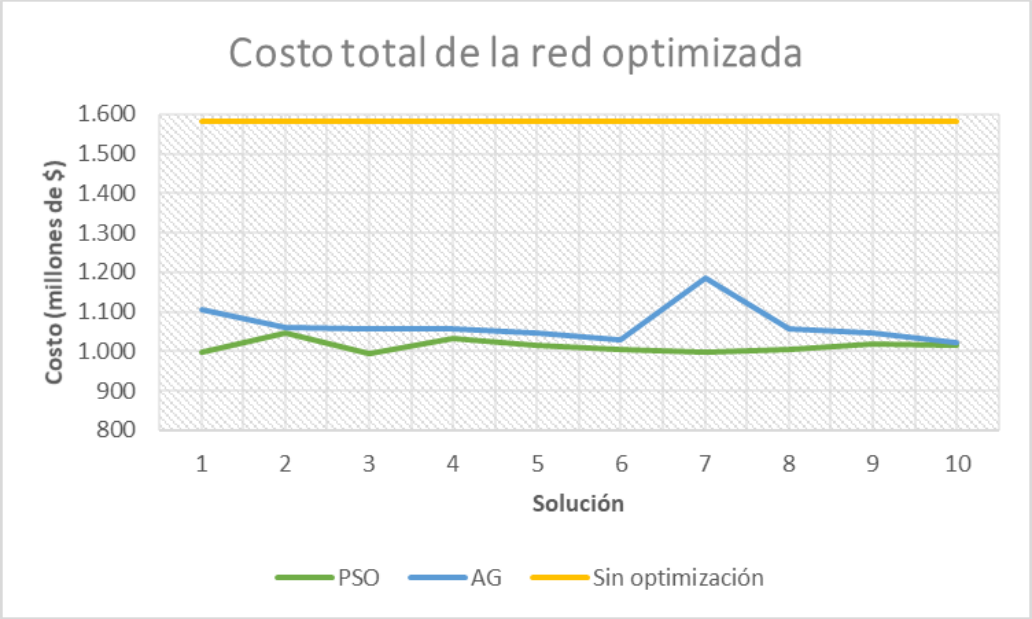


Figura 7. Costo total de la red antes y después de optimizar con AG y PSO.
Fuente: Elaboración propia.

El esfuerzo computacional que conlleva la ejecución de cada uno de los algoritmos de optimización diseñados puede visualizarse al comparar tanto la cantidad de iteraciones que se deben ejecutar para conseguir el costo mínimo como el tiempo que tardan las ejecuciones (Ver figuras 8 y 9).

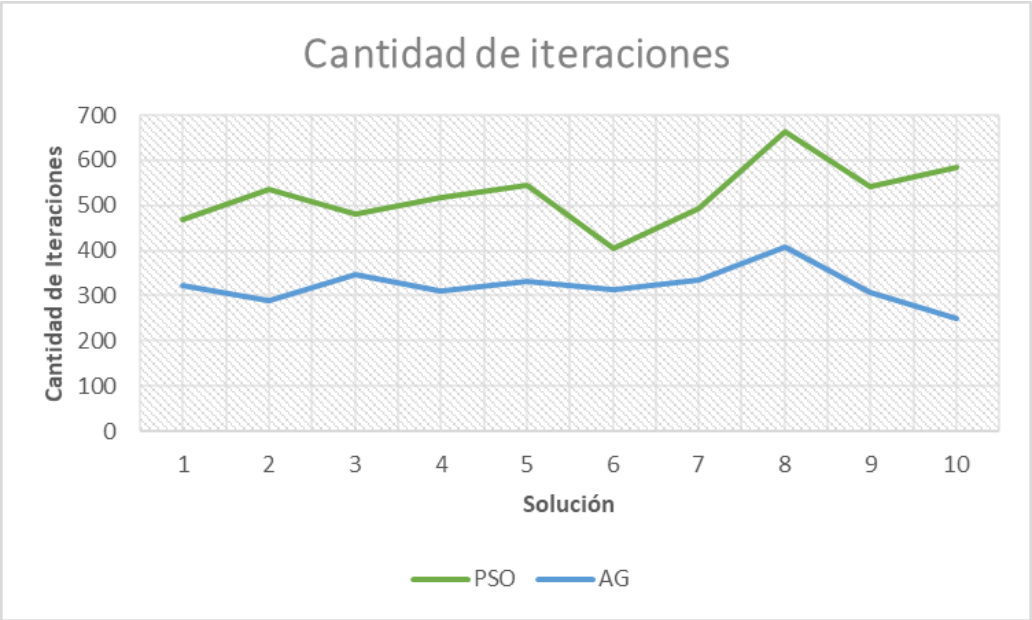


Figura 8. Cantidad de iteraciones ejecutadas para encontrar un costo mínimo de la red optimizada con AG y PSO.

Fuente: Elaboración propia.

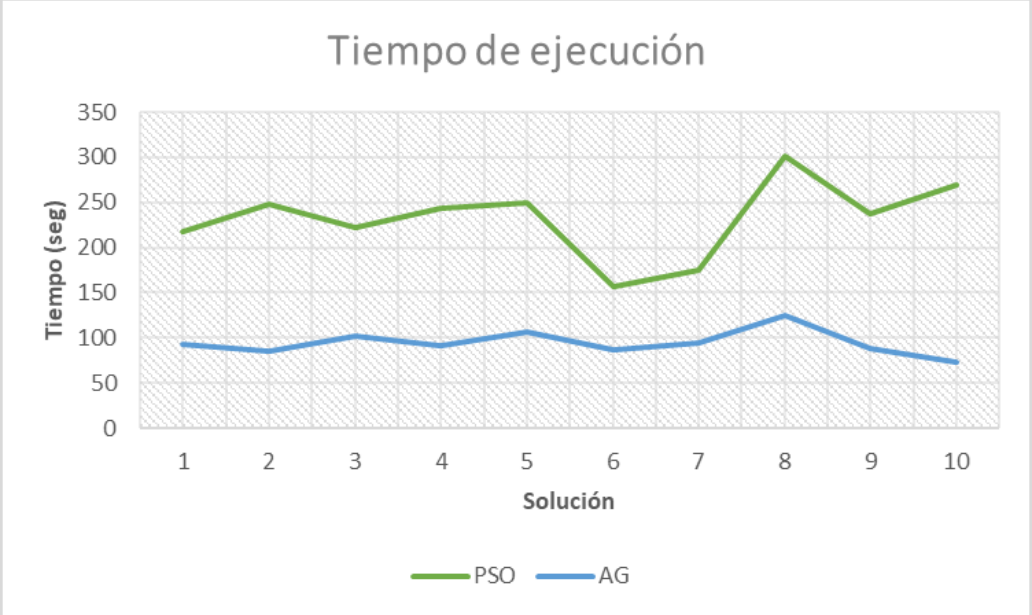


Figura 9. Tiempo de ejecución requerido para encontrar un costo mínimo de la red optimizada con AG y PSO.

Fuente: Elaboración propia.

De las anteriores figuras es fácil inferir que la aplicación del Algoritmo Genético representa un menor esfuerzo computacional, pues el número promedio de iteraciones que requiere para su ejecución es 38,61% más bajo que las que deben ejecutarse al utilizar el algoritmo PSO; en el mismo sentido se puede afirmar que el Algoritmo Genético es 59,34% más rápido en su ejecución que el algoritmo PSO.

Capítulo IV

Aplicación de los Algoritmos Genético y PSO a un ejemplo reportado por la literatura

4.1. Red Hanoi

La red de Hanoi fue presentada por primera vez por Fujiwara & Khang (1990). Es una red compuesta por 3 circuitos básicos, 31 nodos, un embalse y 34 tubos. Todos los nodos se encuentran a la misma elevación y no hay pérdidas menores en las tuberías. En este ejemplo para el cálculo de las pérdidas por fricción se usa la ecuación de Hazen-Williams con un coeficiente Chw=130 para todas las tuberías de la red. La LGH de la fuente es de 100 mca (metros de columna de agua) y la presión mínima requerida es de 30 mca en todos los nodos.

El modelo de la red en formato **.inp* para su tratamiento en EPANET se puede observar en la figura 10.

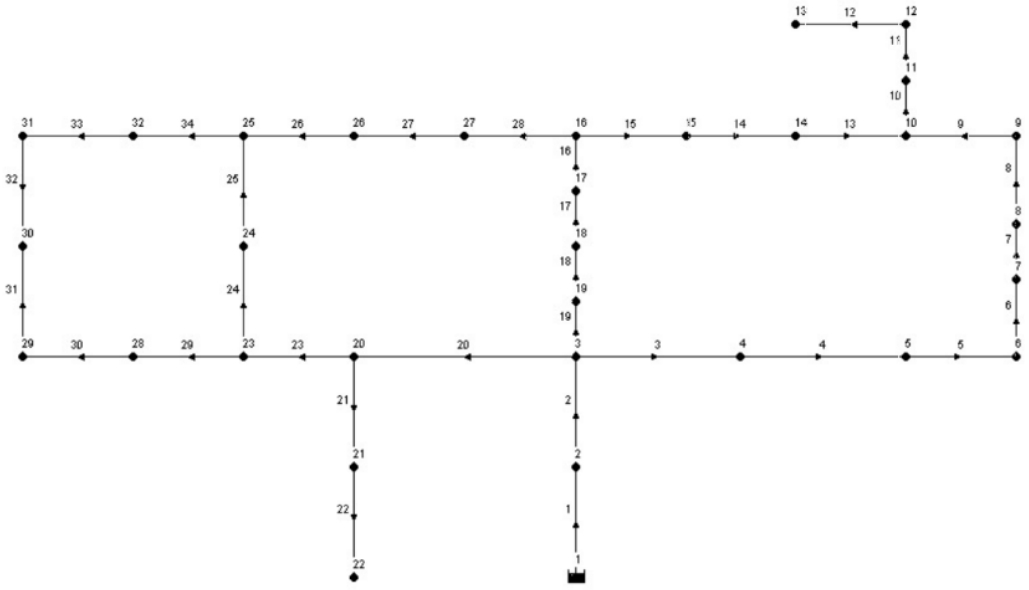


Figura 10. Esquema de conexión de nodos y tuberías Modelo hidráulico red Hanoi. Fuente: Fujiwara & Khang, 1990.

Las tablas 14 y 15 presentan la información de demanda en los nodos y longitud de las tuberías necesaria para construir la red Hanoi.

Tabla 14. Demanda en los nodos de la red Hanoi

ID	Demanda	ID	Demanda	ID	Demanda	ID	Demanda
	m^3/h		m^3/h		m^3/h		m^3/h
2	890	10	525	18	1345	26	900
3	850	11	500	19	60	27	370
4	130	12	560	20	1275	28	290
5	725	13	940	21	930	29	360
6	1005	14	615	22	485	30	360
7	1350	15	280	23	1045	31	105
8	550	16	310	24	820	32	805
9	525	17	865	25	170		

Tabla 15. Longitudes de las tuberías de la red Hanoi

ID	Longitud	ID	Longitud	ID	Longitud	ID	Longitud
	M		m		m		m
1	100	10	950	19	400	28	750
2	1350	11	1200	20	2200	29	1500
3	900	12	3500	21	1500	30	2000
4	1150	13	800	22	500	31	1600
5	1450	14	500	23	2650	32	150
6	450	15	550	24	1230	33	860
7	850	16	2730	25	1300	34	950
8	850	17	1750	26	850		
9	800	18	800	27	300		

El conjunto de diámetros comerciales tomados en cuenta para el diseño de la red Hanoi y sus correspondientes costos por unidad de longitud se muestran en la Tabla 15.

Tabla 16. Diámetros de tubería tomados en cuenta para el diseño de la red Hanoi

Diámetro		Costo
Pulgadas	**Mm**	**US$/m**
12	304,8	45,73
16	406,4	70,4
20	508	98,39
24	609,6	129,33
30	762	180,74
40	1016	278,28

El tamaño del espacio de búsqueda es 6^{34} (aproximadamente $2.86x10^{26}$).

Para evaluar el costo de la red, es necesario determinar mediante una regresión potencial los parámetros k y x que hacen parte de la función de costos que será objeto del proceso de optimización (Ver figura 11).

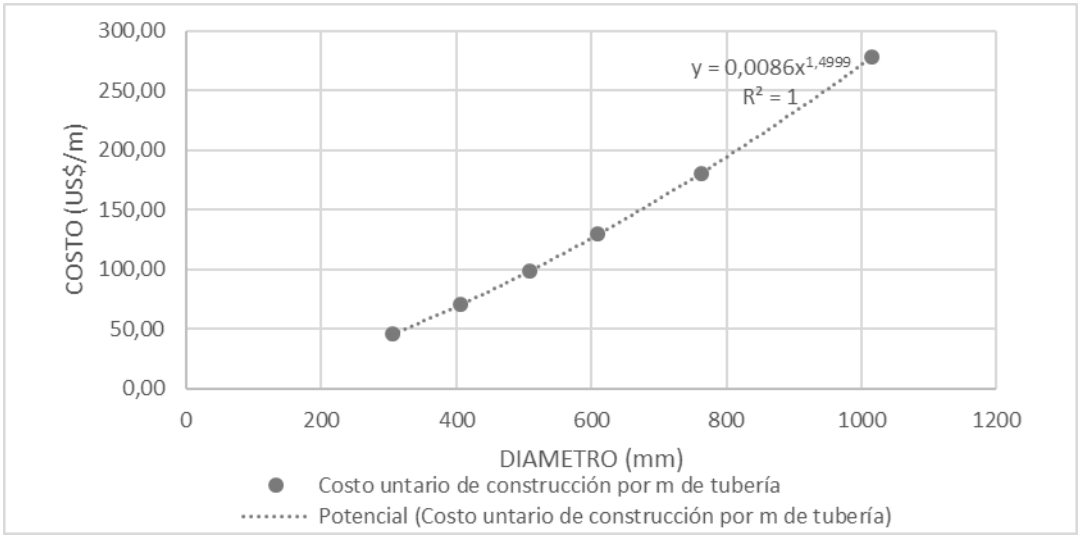


Figura 11. Curva función objetivo de costo para tubería de acuerdo con su diámetro comercial – Red Hanoi.

Fuente: Elaboración propia.

$$k = 0{,}0086$$

$$x = 1{,}4999$$

A continuación, se describe la ecuación de costos objeto del proceso de optimización:

$$C = \sum_{i=1}^{NT} 0{,}0086\, L_i\, D_i^{1{,}4999}$$

4.2. Aplicación del Algoritmo Genético a la red Hanoi

Durante la aplicación del Algoritmo Genético desarrollado, se encontraron diferentes alternativas de solución al diseño de la red Hanoi, las cuales cumplen en su totalidad con la restricción de presión mínima en los nodos. La Tabla 17 resume los costos obtenidos, la cantidad de iteraciones y el tiempo de ejecución necesario para encontrar cada una de las soluciones.

Tabla 17. Soluciones obtenidas aplicando Algoritmo Genético – Red Hanoi

Solución	**Población inicial**	**Costo total de la red optimizada**	**Cantidad de iteraciones**	**Tiempo de ejecución**
Id	**Individuos**	**US $**	**Generaciones**	**seg.**
1	100	6.304.027	276	44,85
2	100	6.649.111	278	48,32
3	100	6.433.142	209	35,34
4	100	6.434.718	531	93,9
5	100	6.667.647	297	49,89
6	100	6.575.386	237	41,42
7	100	6.359.757	358	62,82
8	100	6.642.952	346	57,29
9	100	6.469.510	251	41,53
10	100	6.337.296	313	52,91
Promedio		6.487.355	309,6	52,83

4.2.1. Solución de costo mínimo con AG

Al aplicar el Algoritmo Genético, la solución cuyo valor es mínimo ocurre en la primera ejecución del algoritmo, esta se obtiene tras 44,85 segundos de ejecución y 276 generaciones. A continuación, en la figura 12 se observa la evolución del costo de la red, generación tras generación durante la primera aplicación del algoritmo a la red Hanoi; en las tablas 18, 19 y 20 se resumen los costos de cada tramo de la red Hanoi optimizada y sus principales características hidráulicas; adicionalmente en la figura 13 se representa la presión en los nodos que conforman la red.

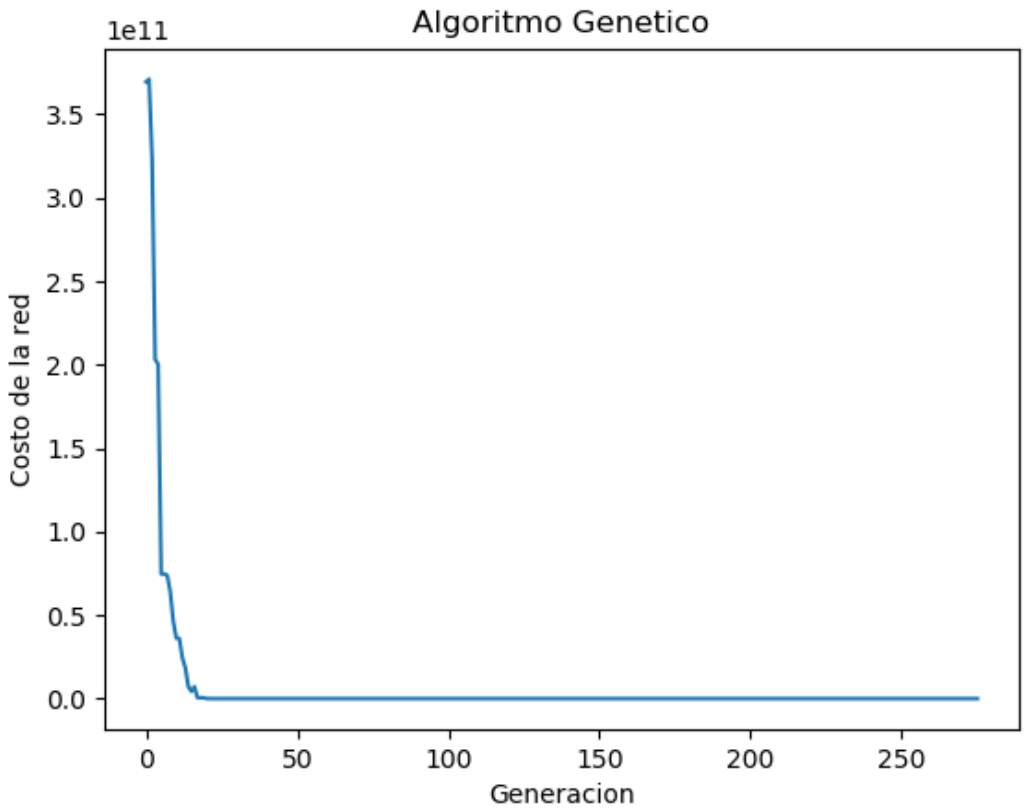


Figura 12. Evolución del costo de la red durante la aplicación del AG a la red Hanoi. Fuente: Elaboración propia.

Tabla 18. Características y costo constructivo de la red Hanoi optimizada con AG

#	Nodos		Long.	Diámetro		Costo
				Nominal	Interno	
	De	A	M	pulg	Mm	US $
1	1	2	100	40	1016	27.832
2	2	3	1350	40	1016	375.727
3	3	4	900	40	1016	250.484
4	4	5	1150	40	1016	320.064
5	5	6	1450	40	1016	403.558
6	6	7	450	40	1016	125.242
7	7	8	850	30	762	153.660
8	8	9	850	40	1016	236.569
9	9	10	800	40	1016	222.653
10	10	11	950	30	762	171.738
11	11	12	1200	30	762	216.932
12	12	13	3500	24	609,6	452.747
13	10	14	800	16	406,4	56.332
14	14	15	500	16	406,4	35.208
15	15	16	550	16	406,4	38.729
16	17	16	2730	16	406,4	192.234
17	17	18	1750	24	609,6	226.373
18	18	19	800	24	609,6	103.485
19	19	3	400	24	609,6	51.742
20	3	20	2200	40	1016	612.295
21	20	21	1500	20	508	147.610
22	21	22	500	16	406,4	35.208
23	20	23	2650	40	1016	737.538
24	23	24	1230	30	762	222.355
25	24	25	1300	24	609,6	168.163
26	26	25	850	20	508	83.645
27	27	26	300	16	406,4	21.125
28	16	27	750	16	406,4	52.812
29	23	28	1500	16	406,4	105.623
30	28	29	2000	16	406,4	140.831
31	29	30	1600	16	406,4	112.665
32	30	31	150	24	609,6	19.403
33	32	31	860	16	406,4	60.557
34	25	32	950	24	609,6	122.888
Costo Total de la Red						6.304.027

Tabla 19. Resultados de la modelación de la red Hanoi optimizada con AG (tuberías)

#	Nodos		Long.	Diámetro		Caudal	Velocidad	Pérd. Unit.
				Nominal	Interno			
	De	A	m	pulg	Mm	LPS	m/s	m/km
1	1	2	100	40	1016	5538,9	6,83	28,59
2	2	3	1350	40	1016	5291,68	6,53	26,27
3	3	4	900	40	1016	2121,04	2,62	4,83
4	4	5	1150	40	1016	2084,93	2,57	4,68
5	5	6	1450	40	1016	1883,54	2,32	3,88
6	6	7	450	40	1016	1604,37	1,98	2,88
7	7	8	850	30	762	1229,37	2,7	7,15
8	8	9	850	40	1016	1076,59	1,33	1,38
9	9	10	800	40	1016	930,76	1,15	1,05
10	10	11	950	30	762	555,56	1,22	1,64
11	11	12	1200	30	762	416,67	0,91	0,96
12	12	13	3500	24	609,6	261,11	0,89	1,2
13	10	14	800	16	406,4	229,37	1,77	6,82
14	14	15	500	16	406,4	58,54	0,45	0,54
15	15	16	550	16	406,4	-19,24	0,15	0,07
16	17	16	2730	16	406,4	191,48	1,48	4,88
17	17	18	1750	24	609,6	-431,76	1,48	3,05
18	18	19	800	24	609,6	-805,37	2,76	9,68
19	19	3	400	24	609,6	-822,04	2,82	10,06
20	3	20	2200	40	1016	2112,5	2,61	4,8
21	20	21	1500	20	508	393,05	1,94	6,23
22	21	22	500	16	406,4	134,72	1,04	2,54
23	20	23	2650	40	1016	1365,28	1,68	2,14
24	23	24	1230	30	762	856,07	1,88	3,66
25	24	25	1300	24	609,6	628,29	2,15	6,11
26	26	25	850	20	508	-266,66	1,32	3,04
27	27	26	300	16	406,4	-16,66	0,13	0,05
28	16	27	750	16	406,4	86,12	0,66	1,11
29	23	28	1500	16	406,4	218,92	1,69	6,25
30	28	29	2000	16	406,4	138,36	1,07	2,67
31	29	30	1600	16	406,4	38,36	0,3	0,25
32	30	31	150	24	609,6	-61,64	0,21	0,08
33	32	31	860	16	406,4	90,81	0,7	1,23
34	25	32	950	24	609,6	314,42	1,08	1,7

Tabla 20. Resultados de la modelación de la red Hanoi optimizada con AG (nodos)

ID Nodo	Demanda	Altura	Presión	
	LPS	m	mca	
2	247,22	97,14	97,14	
3	236,11	61,67	61,67	
4	36,11	57,32	57,32	
5	201,39	51,94	51,94	
6	279,17	46,31	46,31	
7	375	45,02	45,02	
8	152,78	38,94	38,94	
9	145,83	37,77	37,77	
10	145,83	36,93	36,93	
11	138,89	35,37	35,37	
12	155,56	34,22	34,22	
13	261,11	30,01	30,01	
14	170,83	31,48	31,48	
15	77,78	31,21	31,21	
16	86,11	31,25	31,25	
17	240,28	44,56	44,56	
18	373,61	49,9	49,9	
19	16,67	57,65	57,65	
20	354,17	51,12	51,12	
21	258,33	41,77	41,77	
22	134,72	40,5	40,5	
23	290,28	45,45	45,45	
24	227,78	40,96	40,96	
25	47,22	33,01	33,01	
26	250	30,43	30,43	
27	102,78	30,41	30,41	
28	80,56	36,08	36,08	
29	100	30,73	30,73	
30	100	30,33	30,33	
31	29,17	30,35	30,35	
32	223,61	31,4	31,4	
1	-5538,9	100	0	Reservoir

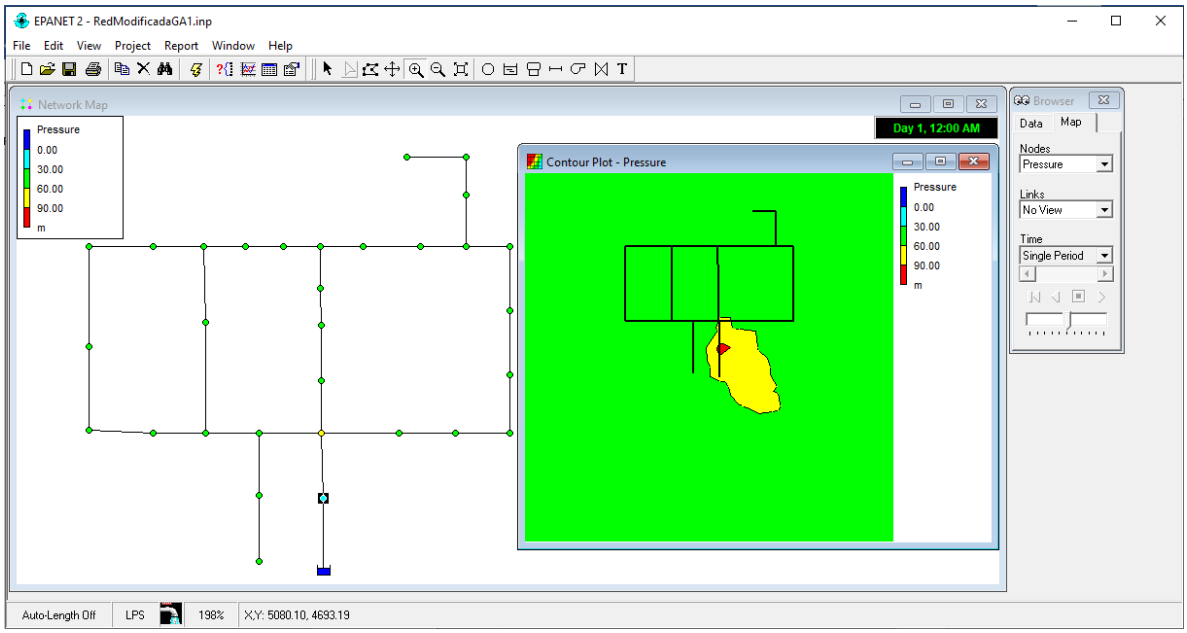


Figura 13. Esquema de la red Hanoi optimizada utilizando AG – Presiones en los nodos.

Fuente: Epanet.

4.3. Aplicación del Algoritmo PSO a la red Hanoi

Al optimizar el diseño de la red Hanoi utilizando el algoritmo PSO desarrollado se encontraron diferentes alternativas de solución, las cuales cumplen en su totalidad con la restricción de presión mínima en los nodos. La Tabla 21 resume los costos obtenidos, la cantidad de iteraciones y el tiempo de ejecución necesario para encontrar cada una de las soluciones.

Tabla 21. Soluciones obtenidas aplicando Algoritmo PSO – Red Hanoi

Solución	**Población inicial**	**Costo total de la red optimizada**	**Cantidad de iteraciones**	**Tiempo de ejecución**
Id	**Partículas**	**US $**	**Iteraciones**	**seg**
1	100	6.803.722	158	46,64
2	100	6.863.934	155	42,93
3	100	7.089.232	541	151,75

Solución	Población inicial	Costo total de la red optimizada	Cantidad de iteraciones	Tiempo de ejecución
Id	**Partículas**	**US $**	**Iteraciones**	**seg**
4	100	7.748.486	149	42,47
5	100	6.395.376	301	86,05
6	100	7.178.569	345	104,05
7	100	7.347.267	265	72,69
8	100	7.132.124	202	58,05
9	100	7.104.698	404	114,06
10	100	6.711.361	665	188,05
Promedio		7.037.477	318,5	90,67

4.3.1. Solución de costo mínimo con PSO

Durante la quinta ejecución del algoritmo PSO, la cual tardo un tiempo estimado de 86,05 segundos y se desarrolló en 301 iteraciones, se obtuvo un costo mínimo de US$ 6.395.376. A continuación, en las tablas 22, 23 y 24 se resumen los costos de cada tramo de la red Hanoi optimizada y sus principales características hidráulicas; adicionalmente en la figura 15 se representa la presión en los nodos que conforman la red.

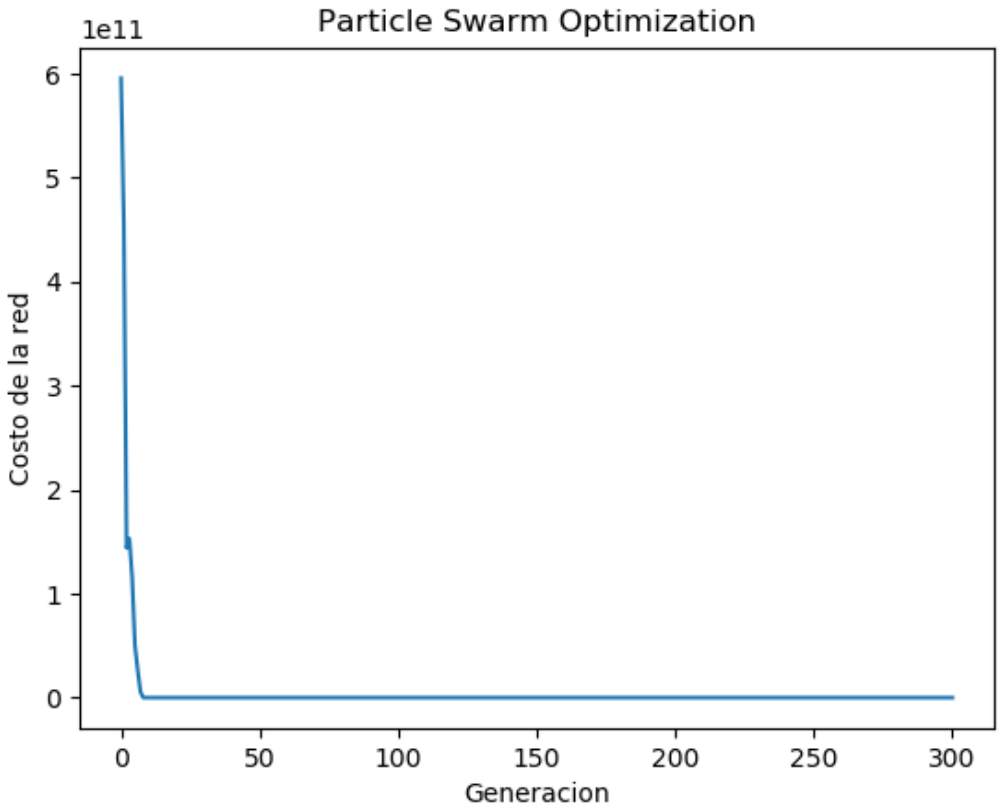


Figura 14. Evolución del costo de la red durante la aplicación del algoritmo PSO a la red Hanoi.

Fuente: Elaboración propia.

Tabla 22. Características y costo constructivo de la red Hanoi optimizada con PSO

#	Nodos		Long.	Diámetro		Costo
				Nominal	Interno	
	De	A	m	pulg	Mm	$
1	1	2	100	40	1016	27.832
2	2	3	1350	40	1016	375.727
3	3	4	900	40	1016	250.484
4	4	5	1150	40	1016	320.064
5	5	6	1450	40	1016	403.558
6	6	7	450	40	1016	125.242
7	7	8	850	40	1016	236.569
8	8	9	850	40	1016	236.569

#	Nodos		Long.	Diámetro		Costo
				Nominal	Interno	
	De	A	m	pulg	Mm	$
9	9	10	800	30	762	144.621
10	10	11	950	30	762	171.738
11	11	12	1200	40	1016	333.979
12	12	13	3500	24	609,6	452.747
13	10	14	800	20	508	78.725
14	14	15	500	16	406,4	35.208
15	15	16	550	12	304,8	25.156
16	17	16	2730	12	304,8	124.863
17	17	18	1750	16	406,4	123.227
18	18	19	800	24	609,6	103.485
19	19	3	400	20	508	39.363
20	3	20	2200	40	1016	612.295
21	20	21	1500	20	508	147.610
22	21	22	500	12	304,8	22.869
23	20	23	2650	40	1016	737.538
24	23	24	1230	40	1016	342.329
25	24	25	1300	30	762	235.010
26	26	25	850	20	508	83.645
27	27	26	300	40	1016	83.495
28	16	27	750	12	304,8	34.303
29	23	28	1500	12	304,8	68.606
30	28	29	2000	12	304,8	91.475
31	29	30	1600	16	406,4	112.665
32	30	31	150	12	304,8	6.861
33	32	31	860	20	508	84.630
34	25	32	950	24	609,6	122.888
Costo Total de la Red						6.395.376

Tabla 23. Resultados de la modelación de la red Hanoi optimizada con PSO (tuberías)

#	Nodos		Long.	Diámetro		Caudal	Velocidad	Pérd. Unit.
				Nominal	Interno			
	De	A	m	pulg	Mm	LPS	m/s	m/km
1	1	2	100	40	1016	5538,9	6,83	28,59
2	2	3	1350	40	1016	5291,68	6,53	26,27
3	3	4	900	40	1016	2183	2,69	5,1
4	4	5	1150	40	1016	2146,89	2,65	4,94

#	Nodos		Long.	Diámetro		Caudal	Velocidad	Pérd. Unit.
				Nominal	Interno			
	De	A	m	pulg	Mm	LPS	m/s	m/km
5	5	6	1450	40	1016	1945,5	2,4	4,12
6	6	7	450	40	1016	1666,33	2,06	3,09
7	7	8	850	40	1016	1291,33	1,59	1,93
8	8	9	850	40	1016	1138,55	1,4	1,53
9	9	10	800	30	762	992,72	2,18	4,81
10	10	11	950	30	762	555,56	1,22	1,64
11	11	12	1200	40	1016	416,67	0,51	0,24
12	12	13	3500	24	609,6	261,11	0,89	1,2
13	10	14	800	20	508	291,33	1,44	3,58
14	14	15	500	16	406,4	120,5	0,93	2,07
15	15	16	550	12	304,8	42,72	0,59	1,23
16	17	16	2730	12	304,8	27,76	0,38	0,55
17	17	18	1750	16	406,4	-268,04	2,07	9,1
18	18	19	800	24	609,6	-641,65	2,2	6,36
19	19	3	400	20	508	-658,32	3,25	16,2
20	3	20	2200	40	1016	2214,25	2,73	5,23
21	20	21	1500	20	508	393,05	1,94	6,23
22	21	22	500	12	304,8	134,72	1,85	10,33
23	20	23	2650	40	1016	1467,03	1,81	2,44
24	23	24	1230	40	1016	1060,39	1,31	1,34
25	24	25	1300	30	762	832,61	1,83	3,47
26	26	25	850	20	508	-368,41	1,82	5,53
27	27	26	300	40	1016	-118,41	0,15	0,02
28	16	27	750	12	304,8	-15,63	0,21	0,19
29	23	28	1500	12	304,8	116,36	1,59	7,87
30	28	29	2000	12	304,8	35,8	0,49	0,89
31	29	30	1600	16	406,4	-64,2	0,49	0,64
32	30	31	150	12	304,8	-164,2	2,25	14,9
33	32	31	860	20	508	193,37	0,95	1,68
34	25	32	950	24	609,6	416,98	1,43	2,86

Tabla 24. Resultados de la modelación de la red Hanoi optimizada con PSO (nodos)

ID Nodo	Demanda	Altura	Presión	
	LPS	m	mca	
2	247,22	97,14	97,14	
3	236,11	61,67	61,67	
4	36,11	57,08	57,08	
5	201,39	51,4	51,4	
6	279,17	45,43	45,43	
7	375	44,04	44,04	
8	152,78	42,4	42,4	
9	145,83	41,1	41,1	
10	145,83	37,25	37,25	
11	138,89	35,69	35,69	
12	155,56	35,41	35,41	
13	261,11	31,2	31,2	
14	170,83	34,39	34,39	
15	77,78	33,35	33,35	
16	86,11	32,68	32,68	
17	240,28	34,19	34,19	
18	373,61	50,11	50,11	
19	16,67	55,19	55,19	
20	354,17	50,16	50,16	
21	258,33	40,81	40,81	
22	134,72	35,64	35,64	
23	290,28	43,69	43,69	
24	227,78	42,04	42,04	
25	47,22	37,53	37,53	
26	250	32,83	32,83	
27	102,78	32,82	32,82	
28	80,56	31,88	31,88	
29	100	30,1	30,1	
30	100	31,13	31,13	
31	29,17	33,37	33,37	
32	223,61	34,81	34,81	
1	-5538,9	100	0	Reservoir

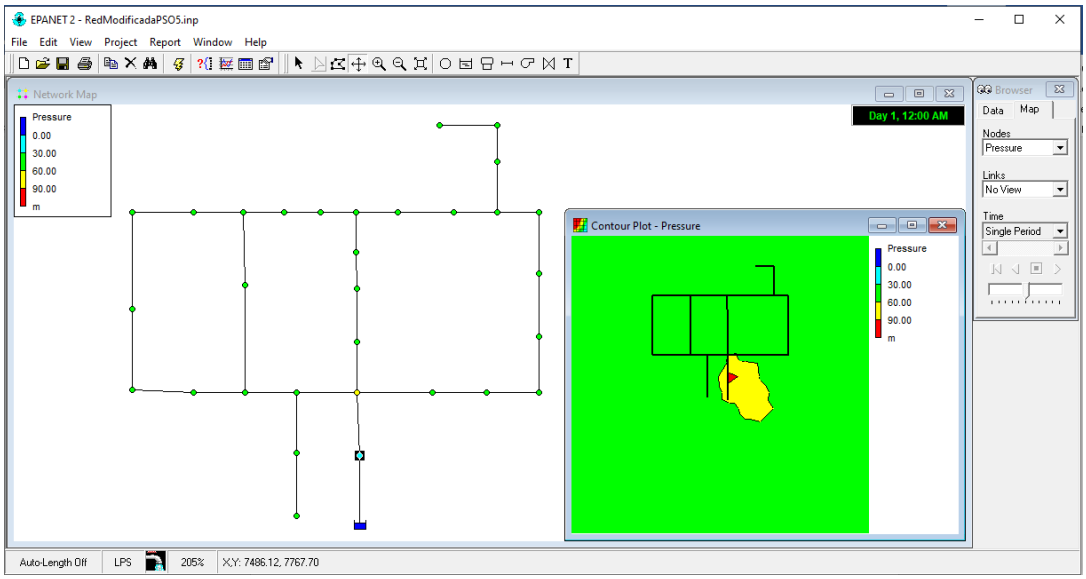


Figura 15. Esquema de la red Hanoi optimizada utilizando PSO – Presiones en los nodos.

Fuente: Epanet.

4.4. Discusión

La comparación de los resultados obtenidos al aplicar tanto el Algoritmo Genético diseñado como el algoritmo PSO en la red Hanoi nos permite observar que, contrario a lo ocurrido al aplicar los algoritmos sobre la red diseñada para el municipio de Mosquera, los costos promedio producto de las optimizaciones ejecutadas con el Algoritmo Genético son 7,82% más bajos que los obtenidos cuando se ejecuta la optimización con PSO.

A continuación, en la figura 16 se observan los costos producto de cada una de las diez optimizaciones ejecutadas utilizando el Algoritmo Genético diseñado y el algoritmo PSO.

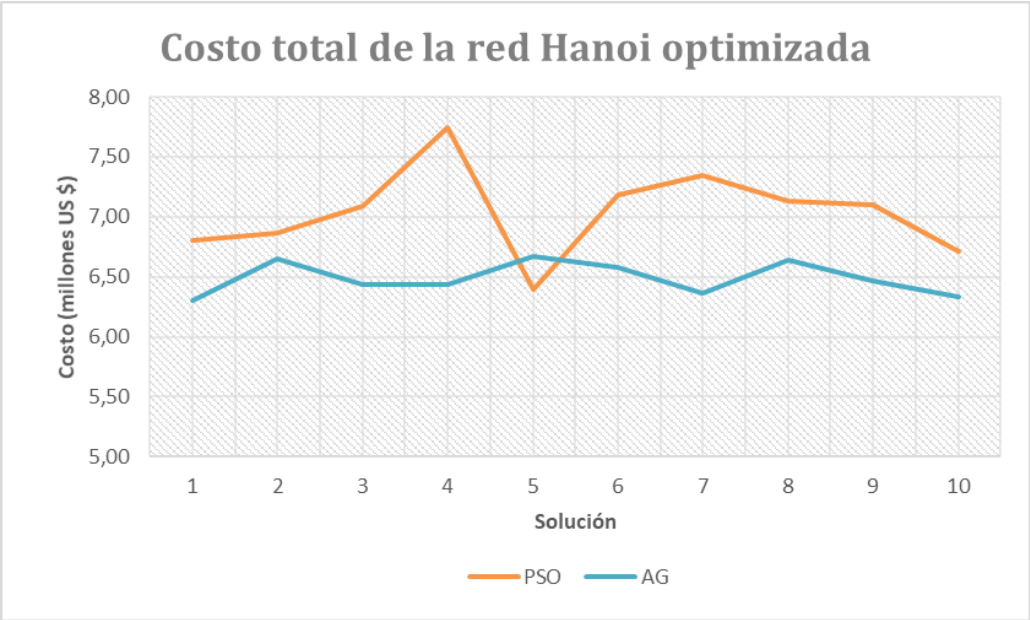


Figura 16. Costo total de la red Hanoi después de optimizar con AG y PSO. Fuente: Elaboración propia.

El esfuerzo computacional que conlleva la ejecución de cada uno de los algoritmos de optimización diseñados puede visualizarse al comparar tanto la cantidad de iteraciones que se deben ejecutar para conseguir el costo mínimo de la red Hanoi como el tiempo que tardan las ejecuciones (Ver figuras 17 y 18).

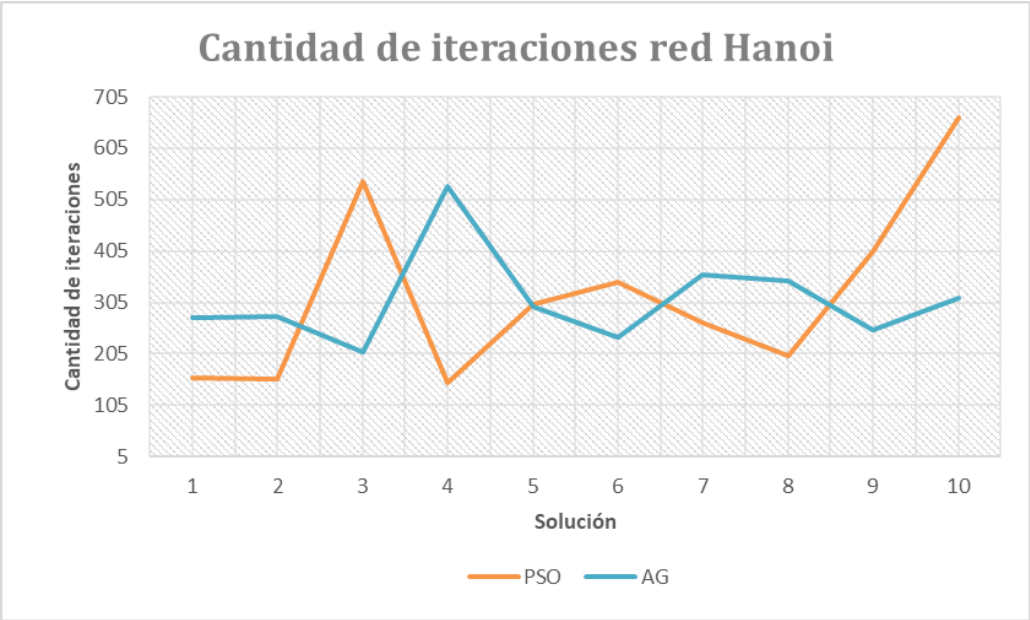


Figura 17. Cantidad de iteraciones ejecutadas para encontrar un costo mínimo de la red Hanoi optimizada con AG y PSO.

Fuente: Elaboración propia.

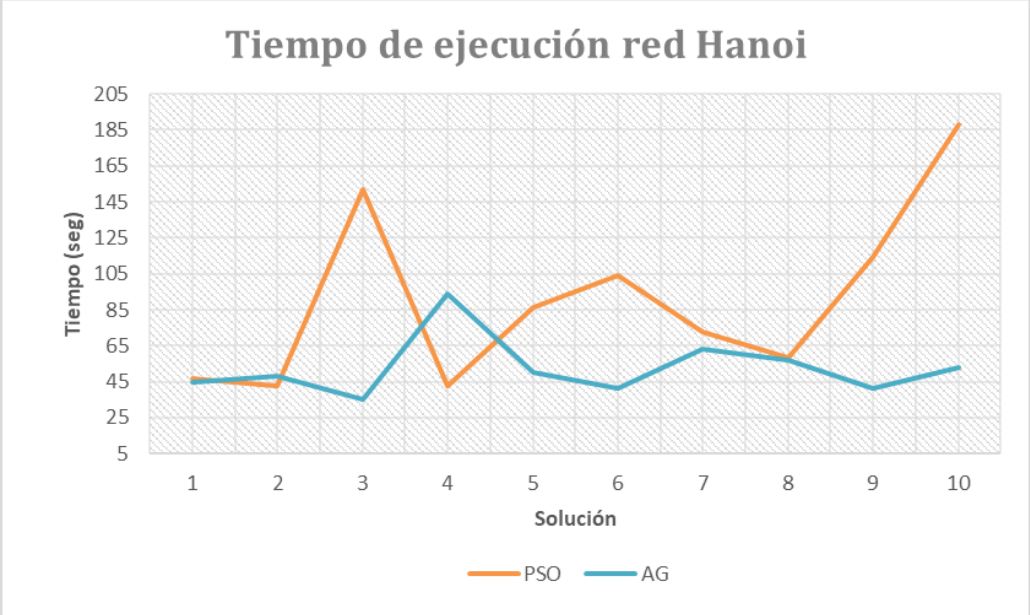


Figura 18. Tiempo de ejecución requerido para encontrar un costo mínimo de la red Hanoi optimizada con AG y PSO.

Fuente: Elaboración propia.

De las anteriores figuras puede inferirse que, para el caso de la red Hanoi al igual que para la red diseñada para el municipio de Mosquera, la aplicación del Algoritmo Genético representa un menor esfuerzo computacional, pues el número promedio de iteraciones que requiere para su ejecución es 2,79% más bajo que las que deben ejecutarse al utilizar el algoritmo PSO; en el mismo sentido se puede afirmar que el Algoritmo Genético es 44,85% más rápido que el algoritmo PSO.

La tabla 25 recoge los resultados obtenidos por distintos investigadores que han trabajado este problema de diseño, así como los resultados obtenidos por medio de los algoritmos estudiados en este trabajo, basados ellos en técnicas heurísticas de diseño.

Tabla 25. Comparación de las soluciones al problema de la red de Hanoi

Autores	**Costo US$**	**Factible**
(Savic & Walters, 1997) GA1	6072412	No
(Savic & Walters, 1997) GA2	6187165	Si
(Cunha & Sousa, 1999) SA	6056163	No
(Wu, Boulos, Orr, & Ro, 2001) fmGA1	6182000	Si
(Liong & Atiquzzaman, 2004) SCE	6224265	Si
AG	6304027	Si
PSO	6395376	Si

Es de aclarar que, en la tabla 25 la columna “Factible” verifica el cumplimiento de la restricción de presión mínima en los nodos (30 mca).

Al comparar los resultados obtenidos, tras la aplicación de los algoritmos Genético y PSO, con los resultados registrados en la literatura se evidencia que, aunque los resultados en sí mismos constituyen buenas soluciones, estos son superiores en un 1,97% el Algoritmo Genético y 3,45 el algoritmo PSO al resultado obtenido por Wu, Boulos, Orr y Ro (2001).

Conclusiones y recomendaciones

Para el desarrollo de la presente investigación se dedujo una ecuación de costos que tuvo en cuenta las tuberías que actualmente ofrece el mercado con sus respectivas características físicas y económicas. Dicha ecuación de costos es fácilmente adaptable al lugar donde se pretenda ejecutar el proyecto, pues deriva de la aplicación de una regresión potencial ejecutada teniendo en cuenta los diámetros internos reales de cada tubería y su costo por metro lineal.

Los algoritmos desarrollados en Python para el diseño óptimo de redes de distribución de agua son adecuados ya que la elección de los diámetros cumple con las restricciones hidráulicas, es decir, garantiza presiones y velocidades adecuadas de acuerdo con la normatividad vigente, el comportamiento energético, la conservación de masa en los nudos y las restricciones comerciales de disponibilidad de diámetros. Estos algoritmos pueden aplicarse a redes de distribución de agua de diversos tamaños, minimizando el problema de diseño y reduciendo los cálculos y tiempos de ejecución.

Los algoritmos diseñados, AG y PSO, se aplicaron satisfactoriamente en una red real ubicada en la sabana de Bogotá, cuyo modelo hidráulico fue desarrollado en el año 2014 como parte del proyecto: “Diseño para el abastecimiento de agua potable (sectorización) del sistema de acueducto para grandes desarrollos de vivienda e industriales del municipio de Mosquera Departamento Cundinamarca”.

Los resultados obtenidos al aplicar los algoritmos desarrollados en el modelo *Sectorizacion_2024.inp* permiten concluir que, aunque cualquiera de los dos algoritmos reduce considerablemente los costos de la red inicialmente diseñada (reducción promedio PSO = 35,99%; reducción promedio AG = 32,50%), con el algoritmo PSO se obtienen costos inferiores a los obtenidos con el Algoritmo Genético. De otra parte, el Algoritmo Genético presenta un menor esfuerzo computacional, pues el número promedio de iteraciones que requiere para su ejecución es 38,61% más bajo que las requeridas en el algoritmo PSO; en el

mismo sentido se puede afirmar que el Algoritmo Genético es 59,34% más rápido en su ejecución que el algoritmo PSO.

Los algoritmos diseñados también se aplicaron a una red de suministro de agua ampliamente estudiada en la literatura, pero contrario a lo sucedido con la red ubicada en la sabana de Bogotá, los resultados de las optimizaciones aplicadas a la red Hanoi arrojaron que el Algoritmo Genético proporciona costos de la red inferiores a los obtenidos con PSO. En general el Algoritmo Genético para este caso también presenta menor esfuerzo computacional.

Aunque la comparación de los costos mínimos obtenidos con la aplicación de los Algoritmos Genético y PSO diseñados en esta investigación con los costos mínimos reportados por la literatura para la Red Hanoi, muestra que los resultados de estos algoritmos arrojan costos ligeramente superiores, puede concluirse que en sí constituyen buenas soluciones, pues con un mínimo esfuerzo computacional se obtienen resultados que, en el mejor de los casos, solo superan en un 1,97% el mínimo reportado por la literatura.

Los algoritmos desarrollados en esta investigación son de baja complejidad lo que implica un pequeño número de simulaciones hidráulicas y por lo tanto una excelente velocidad de ejecución.

Con las metodologías propuestas se llega a una buena aproximación para superar el problema de diseño de redes de distribución de agua potable de costo mínimo.

Para mejorar la efectividad del Algoritmo Genético diseñado se recomienda realizar simulaciones con distintas probabilidades de cruce y mutación, a fin de analizar la influencia que ejerce cada uno de estos operadores en la solución final que proporciona el algoritmo. Del mismo modo podría analizarse la relevancia de los límites de velocidad impuestos al desplazamiento de cada partícula en el algoritmo PSO.

Aunque, los códigos escritos en Python que hacen parte de la presente investigación se desarrollaron con el único fin de optimizar el diseño de una red de acueducto realmente construida en la Sabana de Bogotá, es evidente que a futuro estos podrían constituir la base de un programa mucho más amplio que permitiese de manera sencilla generar diseños óptimos de redes de acueducto de diferentes tamaños y configuraciones utilizando, además de las técnicas aquí programadas (Algoritmo Genético – AG y Particle Swarm Optimization - PSO); otras que también se encuentran contenidas dentro de la librería DEAP y se basan en: Programación Genética (GP), Estrategias de Evolución (ES) y Estimaciones de Algoritmos de Distribución (EDA).

Con el fin de identificar los limites reales de las aplicaciones desarrolladas, se recomienda realizar pruebas con modelos de redes de distribución de agua cuya cantidad de nodos y tramos de tuberías sea superior a los que presentan las redes aquí estudiadas.

Para la construcción de un programa que permita realizar diseños de redes de distribución de agua ajustados a la realidad, se recomienda complementar los códigos desarrollados en esta investigación incluyendo en ellos una restricción que impida que la diferencia de diámetros entre dos tramos consecutivos sea superior a dos diámetros comerciales, esto con el fin de garantizar que en el mercado existan los accesorios necesarios para realizar las transiciones.

Es necesario analizar la respuesta y los resultados de los algoritmos desarrollados al aplicarlos a modelos de redes de distribución de agua que contengan diferentes tipos de accesorios y que por tanto presenten perdidas menores localizadas.

La experiencia del usuario de un futuro programa que se cree a partir de los códigos aquí desarrollados, podría mejorar sí los mismos códigos se modifican de tal manera que al correr el programa se le facilite al usuario ingresar las restricciones que desee imponer al diseño (velocidades y presiones mínimas y máximas) y cambiar los tipos de tubería y sus precios por metro lineal.

Bibliografía

Alperovits, E., & Shamir, U. (1977). Design of optimal water distribution systems. *Water Resources Research*, *13*(6), 885–900. https://doi.org/10.1029/WR013i006p00885

Baños, R., Gil, C., Reca, J., & Montoya, F. G. (2010). A memetic algorithm applied to the design of water distribution networks. *Applied Soft Computing Journal*, *10*(1), 261–266. https://doi.org/10.1016/j.asoc.2009.07.010

Collette, Y., Hansen, N., Pujol, G., Salazar Aponte, D., & Le Riche, R. (2010). Object-Oriented Programming of Optimizers - Examples in Scilab. In R. F. Coelho & P. Breitkopf (Eds.), *Multidisciplinary Design Optimization in Computational Mechanics* (pp. 527–565). https://doi.org/10.1002/9781118600153.ch14

Cunha, M. da C., & Ribeiro, L. (2004). Tabu search algorithms for water network optimization. *European Journal of Operational Research*, *157*(3), 746–758. https://doi.org/10.1016/S0377-2217(03)00242-X

Cunha, M. da C., & Sousa, J. (1999). Water Distribution Network Design Optimization: Simulated Annealing Approach. *Journal of Water Resources Planning and Management*, *125*(4), 215–221. https://doi.org/doi.org/10.1061/(ASCE)0733-9496(1999)125:4(215)

Dandy, G. C., Simpson, A. R., & Murphy, L. J. (1996). An improved genetic algorithm for pipe network optimization. *Water Resources Research*, *32*(2), 449–458. https://doi.org/10.1029/95WR02917

Dardani, I., & Jones, G. (2018). Algorithms for optimization of branching gravity-driven water networks. *Drinking Water Engineering and Science*, *11*(1), 67–85. https://doi.org/10.5194/dwes-11-67-2018

De Rainville, F.-M., Fortin, F.-A., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: A Python Framework for Evolutionary Algorithms. *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference Companion - GECCO Companion '12*, 85. https://doi.org/10.1145/2330784.2330799

Eusuff, M. M., & Lansey, K. E. (2003). Optimization of Water Distribution Network

Design Using the Shuffled Frog Leaping Algorithm. *Journal of Water Resources Planning and Management*, *129*(3), 210–225. https://doi.org/10.1061/(asce)0733-9496(2003)129:3(210)

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, *13*, 2171–2175.

Fujiwara, O., & Khang, D. B. (1990). A Two-Phase Decomposition Method for Optimal Design of Looped Water Distribution Networks. *Water Resources Research*, *26*(4), 539–549. https://doi.org/doi.org/10.1029/WR026i004p00539

Geem, Z. W. (2006). Optimal cost design of water distribution networks using a decomposition approach. *Engineering Optimization*, *38*(3), 259–277. https://doi.org/10.1080/0305215X.2016.1157689

Iglesias, P. L., Mora, D., Lopez, P. A., & García, J. (2009). Aplicación de métodos heurísticos al diseño de redes de agua. *Revista Ingeniería Del Agua*, (1).

Integrita S.A.S. (2014). *Diseño para la abastecimiento de agua potable (sectorización) del sistema de acueducto para grandes desarrollos de vivienda e industriales del municipio de Mosquera Departamento Cundinamarca*. 56.

Kadu, M. S., Gupta, R., & Bhave, P. R. (2008). Optimal Design of Water Networks Using a Modified Genetic. *Journal of Water Resources Planning and Management*, *134*(2), 147–160. https://doi.org/doi:10.1061/(ASCE)0733-9496(2008)134:2(147)

Keedwell, E., & Khu, S.-T. (2006). Novel cellular automata approach to optimal water distribution network design. *Journal of Computing in Civil Engineering*, *20*(1), 49–56. https://doi.org/10.1061/(ASCE)0887-3801(2006)20:1(49)

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, *IV*, 1942–1948. https://doi.org/10.1109/ICNN.1995.488968

Liong, S., & Atiquzzaman, M. (2004). Optimal design of water distribution network using shuffled complex evolution. *Journal of The Institution of Engineers, Singapore*, *44*(1), 93–107.

Lopez, L., Cuero, P., Díaz, O., Páez, D., & Saldarriaga, J. G. (2013). Diseño de redes de

distribución de agua potable por medio de la metodología OPUS e inicio en caliente. *XII Simposio Iberoamericano Sobre Planificación de Sistemas de Abastecimiento y Drenaje*, (1), 12.

Maier, H. R., Simpson, A. R., Zecchin, A. C., Foong, W. K., Phang, K. Y., Seah, H. Y., & Tan, C. L. (2003). Ant Colony Optimization for the Design of Water Distribution Systems. *Journal of Water Resources Planning and Management*, *129*(3), 200–209. https://doi.org/10.1061/(ASCE)0733-9496(2003)129:3(200)

Méndez, M. (2014). Diseño óptimo de un sistema de distribución de agua (SDA) aplicando el algoritmo Simulated Annealing (SA). *Revista Tecnología En Marcha*, *27*(3), 22–31. Retrieved from http://revistas.tec.ac.cr/index.php/tec_marcha/article/view/2063

Ministerio de Vivienda Ciudad y Territorio. *Resolucion 0330*. , (2017).

Mohan, S., & Babu, K. S. J. (2010). Optimal Water Distribution Network Design with Honey-Bee Mating Optimization. *Journal of Computing in Civil Engineering*, *24*(1), 117–126. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000018

Montalvo, I. (2008). Diseño Óptimo de Sistemas de Distribución de Agua Mediante Particle Swarm (Universidad Politecnica de Valencia). Retrieved from https://riunet.upv.es/handle/10251/14858

Montesinos, P., García-Guzmán, A., & Ayuso, J. (1997). Optimización De Redes De Distribución De Agua Utilizando Un Algoritmo Genético. *Revista Ingeniería Del Agua*, *4*(26), 71–77. https://doi.org/10.4995/ia.1997.2717

Mora, D., Iglesias, P. L., Martinez, F. J., & Ballesteros, P. (2015). Efficiency of Evolutionary Algorithms in Water Network Pipe Sizing. *Water Resources Management*, *29*(13), 4817–4831. https://doi.org/10.1007/s11269-015-1092-x

Mora, D., Iglesias, P. L., Martinez, F. J., & Fuertes, V. S. (2013). Design of Water Distribution Networks using a Pseudo-Genetic Algorithm and Sensitivity of Genetic Operators. *Water Resources Management*, *27*(12), 4149–4162. https://doi.org/10.1007/s11269-013-0400-6

Mora, Daniel. (2012). Diseño de redes de distribución de agua mediante algoritmos evolutivos. Análisis de eficiencia (Universidad Politécnica de Valencia). Retrieved from https://riunet.upv.es/handle/10251/16803

Morgan, D. R., & Goulter, I. C. (1985). Optimal urban water distribution design. *Water Resources Research*, *21*(5), 642–652. https://doi.org/10.1029 / WR021i005p00642

Nárvaez, P., & Galeano, H. (2002). Ecuación de costos y función objetivo para la optimización del diseño de redes de flujo de líquidos a presión. *Ingeniería e Investigación*, (49), 23–29. Retrieved from https://revistas.unal.edu.co/index.php/ingeinv/article/view/21407

Pereyra, G., Pandolfi, D., & Villagra, N. (2017). Diseño y optimización de redes de distribución de agua utilizando algoritmos genéticos. *Revista de Informes Científico Técnicos de La Universidad Nacional de La Patagonia Austral*, *9*(1), 37–63. Retrieved from https://dialnet.unirioja.es/servlet/articulo?codigo=5919083

Pino, E., Valle, A., Condori, F., Mejia, J., Chavarri, E., & Alfaro, L. (2017). Diseño Óptimo de Redes de Distribución de Agua Usando Un Software Basado En Microalgoritmos Genéticos Multiobjetivos. *Ribagua*, *4*(1), 6–23. https://doi.org/10.1080/23863781.2017.1317087

Reca, J., & Martínez, J. (2006). Genetic algorithms for the design of looped irrigation water distribution networks. *Water Resources Research*, *42*(5), 1–9. https://doi.org/10.1029/2005WR004383

Reed, J., Toombs, R., & Barricelli, N. A. (1967). Simulation of biological evolution and machine learning. I. Selection of self-reproducing numeric patterns by data processing machines, effects of hereditary control, mutation type and crossing. *Journal of Theoretical Biology*, *17*(3), 319–342. https://doi.org/10.1016/0022-5193(67)90097-5

Rincón, J. C. (2006). Aplicación de algoritmos genéticos en la optimización del sistema de abastecimiento de agua Barquisimeto-Cabudare. *Avances En Recursos Hidráulicos*, *14*(10), 25–38. Retrieved from https://revistas.unal.edu.co/index.php/arh/article/view/9328

Rodríguez, K., Fuentes, Ó. A., Jiménez, M. R., & Cruz, F. de L. (2006). Diseño óptimo de redes de distribución de agua potable utilizando un algoritmo genético multiobjetivo. *Seminário Iberoamericano Sobre Sistemas de Abastecimento Urbano de Água João Pessoa*, 10.

Rojas, F. (2014). Políticas e institucionalidad en materia de agua potable y saneamiento en

América Latina y el Caribe. *Naciones Unidas, CEPAL, Serie Recursos Naturales e Infraestructura*, *166*(3), 1–79. https://doi.org/10.3989/arbor.2000.i653.1000

Rossman, L. A. (2000). *Epanet 2 Users Manual*. https://doi.org/10.1177/0306312708089715

Saldarriaga, J., & Mendoza, F. L. (2010). Diseño optimizado de Redes de Distribución de Agua Potable Incluyendo Análisis de Costo Mínimo versus Resiliencia de la Red. *XXIV Congreso Latinoamericano de Hidráulica Selección de Trabajos Punta Del Este, Uruguay, 2010*.

Sanvicente Sanchez, H., & Solís, J. F. (2003). Optimización de los diámetros de las tuberías de una red de distribución de agua mediante algoritmos de recocido simulado. *Tecnología y Ciencias Del Agua*, *18*(1), 105–118.

Savic, D. A., & Walters, G. A. (1997). Genetic Algorithms for Least-Cost Design of Water. *Journal of Water Resources Planning and Management*, *123*(2), 67–77. https://doi.org/10.1061/(ASCE)0733-9496(1997)123

Simpson, A. R., Dandy, G. C., & Murphy, L. J. (1994). Genetic Algorithms Compared to Other Techniques for Pipe Optimization. *Water Resources Planning and Management*, *120*(4), 423–443. https://doi.org/10.1061/(ASCE)0733-9496(1994)120:4(423)

Sung, Y.-H., Lin, M.-D., Lin, Y.-H., & Liu, Y.-L. (2007). Tabu Search Solution of Water Distribution Network Optimization. *Journal of Environmental Engineering and Management*, *17*(3), 177–187.

Suribabu, C. R. (2010). Differential evolution algorithm for optimal design of water distribution networks. *Journal of Hydroinformatics*, *12*(1), 66. https://doi.org/10.2166/hydro.2010.014

Suribabu, C. R., & Neelakantan, T. R. (2006). Design of water distribution networks using particle swarm optimization. *Urban Water Journal*, *3*(2), 111–120. https://doi.org/10.1080/15730620600855928

Tolson, B. A., Maier, H. R., Simpson, A. R., & Lence, B. J. (2004). Genetic algorithms for reliability-based optimization of water distribution systems. *Journal of Water Resources Planning and Management*, *130*(1), 63–72. https://doi.org/10.1061/(ASCE)0733-9496(2004)130:1(63)

Vairavamoorthy, K., & Ali, M. (2000). Optimal design of water distribution systems using genetic algorithms. *Computer-Aided Civil and Infrastructure Engineering*, *15*(5), 374–382. https://doi.org/doi.org/10.1111/0885-9507.00201

Vargas Cordero, Z. R. (2009). La investigación aplicada: una forma de conocer las realidades con evidencia científica. *Revista Educación*, *33 (1)*(0379–7082), 155–165. Retrieved from https://www.redalyc.org/pdf/440/44015082010.pdf

Vasan, A., & Simonovic, S. P. (2010). Optimization of Water Distribution Network Design Using Differential Evolution. *Journal of Water Resources Planning and Management*, *136*(2), 279–287. https://doi.org/10.1061/(ASCE)0733-9496(2010)136:2(279)

Vegas, O., Martínez, F., Alonso, J., & Tzatchkov, V. (2017). *Iniciación a la Programación con la Toolkit de Epanet v2.00.12 en un Entorno Windows*.

Wu, Z. Y., Boulos, P. F., Orr, C. H., & Ro, J. J. (2001). Using Genetic Algorithms to Rehabilitate Distribution Systems. *Journal - American Water Works Association*, *93*(11), 74–85. https://doi.org/10.1002/j.1551-8833.2001.tb09335.x

Yates, D. F., Templeman, A. B., & Boffey, T. B. (1984). The computational complexity of the problem of determining least capital cost designs for water supply networks. *Engineering Optimization*, *7*, 143–155. https://doi.org/https://doi.org/10.1080/03052158408960635

Zheng, F., Zecchin, A. C., Simpson, A. R., & Lambert, M. F. (2014). Noncrossover Dither Creeping Mutation-Based Genetic Algorithm for Pipe Network Optimization. *Journal of Water Resources Planning and Management*, *140*(4), 553–557. https://doi.org/10.1061/(ASCE)WR.1943-5452.0000351

Anexos

Anexo 1. Código función de evaluación y restricciones

```
# TESIS DE GRADO
# OPTIMIZACION DEL DISEÑO DE UNA RED DE DISTRIBUCION DE AGUA POTABLE
# ECUACIÓN DE EVALUACION (COSTOS) Y RESTRICCIONES
# CREADO POR:  Henry Omar Peñaloza Mantilla
# ECI



#importar dependencias
import numpy as np
import os

from  epanettools.epanettools import EPANetSimulation, Node, Link, Network, Nodes, Links, Patterns, Controls, Control
from epanettools import epanet2 as epa
from os import remove

def evalfit2(features,es):

    diametros=Link.value_type['EN_DIAMETER']
    longitudes=Link.value_type['EN_LENGTH']
    velocidades=Link.value_type['EN_VELOCITY']
    presiones=Node.value_type['EN_PRESSURE']

    cant_link=es.network.links #cantidad de tubos
    cant_node=es.network.nodes #cantidad de nodos
```

```
    VA=[]
    PEN1=[]
    PEN2=[]

    for id in range (len(cant_link)):
        r=(es.ENsetlinkvalue((id+1),diametros,features[id]))
        f=os.path.join('parcial_modificada.inp')

a=((es.ENgetlinkvalue(id+1,longitudes)[1])*(1.4078*(es.ENgetlinkvalue(id+1,diametros)[1
])**2.0707))
        VA.append(a)
    cost = np.sum(VA)

    es.ENsolveH();

    for id in range (1,len(cant_node)):
        if (es.ENgetnodevalue(id,presiones)[1]) < 20:
            pen_pre = (20-(es.ENgetnodevalue(id,presiones)[1]))*100000000
        elif (es.ENgetnodevalue(id,presiones)[1]) > 50:
            pen_pre = ((es.ENgetnodevalue(id,presiones)[1])-50)*100000000
        else:
            pen_pre = 0
        PEN1.append(pen_pre)
    PP=np.sum(PEN1)

    for id in range (1,len(cant_link)):
        if (es.ENgetlinkvalue(id,velocidades)[1]) < 0.5:
            pen_vel = (0.5-(es.ENgetlinkvalue(id,velocidades)[1]))*100000000
        elif (es.ENgetlinkvalue(id,velocidades)[1]) > 2.5:
```

```
        pen_vel = ((es.ENgetlinkvalue(id,velocidades)[1])-2.5)*100000000
    else:
        pen_vel = 0
    PEN2.append(pen_vel)
PV=np.sum(PEN2)

costo=round(cost+PP+PV)

return costo,
remove('parcial_modificada.inp')
```

Anexo 2. Código Algoritmo Genético

```
# TESIS DE GRADO
# OPTIMIZACION DEL DISEÑO DE UNA RED DE DISTRIBUCION DE AGUA
POTABLE
# ALGORITMO GENETICO
# CREADO POR:  Henry Omar Peñaloza Mantilla
# ECI



#importar dependencias
import pandas as pd
import numpy as np
import os
import random
import matplotlib.pyplot as plt
import pyfiglet

import EC_EVAL

from epanettools.epanettools import EPANetSimulation, Node, Link, Network,
Nodes,Links, Patterns, Controls, Control
from time import time
from deap import creator, base, tools, algorithms

pd.options.display.float_format ='{:.2f}'.format

ascii_banner = pyfiglet.figlet_format("Genetic Algorithm GA")
print(ascii_banner)
```

```
#Ruta de acceso Red original
archivo = str(input("Ingrese el nombre del archivo a optimizar (incluya la extenson.inp): "))

path_Inp = archivo;

es=EPANetSimulation(path_Inp)

diametros=Link.value_type['EN_DIAMETER']
longitudes=Link.value_type['EN_LENGTH']
caudales=Link.value_type['EN_FLOW']
velocidades=Link.value_type['EN_VELOCITY']
presiones=Node.value_type['EN_PRESSURE']

cant_link=es.network.links #cantidad de tubos
cant_node=es.network.nodes #cantidad de nodos

L=[] #longitud
V=[] #velocidad del flujo
D=[] #diametro inicial
D_new=[] #nuevo diametro escogido
VA=[] #valor por tramo


for id in range (1,len(cant_link)+1):
   D.append(es.ENgetlinkvalue(id,diametros)[1])

a=round((es.ENgetlinkvalue(id,longitudes)[1])*(1.4078*(es.ENgetlinkvalue(id,diametros)[
1])**2.0707))
   VA.append(a)
```

```
lista_links=list(es.network.links)[:len(cant_link)+1]
id_links=[es.network.links[x].id for x in lista_links]

### Mostrar Resultados Iniciales####
tabla_d= pd.DataFrame(np.array(D).reshape(len(cant_link),1), columns = ['Diametro
Inicial'])
tabla_va= pd.DataFrame(np.array(VA).reshape(len(cant_link),1), columns = ['Valor
inicial'])

array = np.array(VA)
salida = np.sum(array)

DiametrosComerciales=(82.04, 105.52, 155.32, 202.17, 252.07, 298.95, 328.26, 375.16,
422.04, 468.94, 562.72)

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()

DIM=len (cant_link)

def evalfit(individual):
    return EC_EVAL.evalfit2(features=individual,es=es)

# Inicializadores
toolbox.register("diam", random.choice, DiametrosComerciales)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.diam, n=DIM)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
```

```
toolbox.register("evaluate", evalfit)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

#----------

def main():

    poblacion = int(input("Ingrese cantidad de individuos - poblacion inicial: "))
    pop = toolbox.population(n=poblacion)

    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("media", np.mean)
    stats.register("desv est", np.std)
    stats.register("min", np.min)
    stats.register("max", np.max)

    logbook = tools.Logbook()
    logbook.header = ["gen", "evals"] + stats.fields

    CXPB = float(input("Ingrese la probabilidad de cruce: "))
    MUTPB = float(input("Ingrese la probabilidad de mutacion: "))

    print("Inicio de la evolucion")

    # Evaluar a toda la población
    fitnesses = list(map(toolbox.evaluate, pop))
    for ind, fit in zip(pop, fitnesses):
        ind.fitness.values = fit
```

```
# Extrayendo todos los fitness de
fits = [ind.fitness.values[0] for ind in pop]

# Seguimiento variable del número de generaciones
g = 0

# Variable - Criterio de parada
n=0

MIN=[]

tiempo_inicial = time()

# Comienza la evolución
while n < 100:

    # Una nueva generación
    g = g + 1

    # Seleccione los individuos de la siguiente generación
    offspring = toolbox.select(pop, len(pop))
    # Clonar los individuos seleccionados
    offspring = list(map(toolbox.clone, offspring))

    # Aplicar cruce y mutación en la descendencia
    for child1, child2 in zip(offspring[::2], offspring[1::2]):

        # cruzar dos individuos con probabilidad CXPB
        if random.random() < CXPB:
```

```
        toolbox.mate(child1, child2)

        # valores de condición física de los niños, debe recalcularse más tarde
        del child1.fitness.values
        del child2.fitness.values

for mutant in offspring:

    # mutar un individuo con probabilidad MUTPB
    if random.random() < MUTPB:
        toolbox.mutate(mutant)
        del mutant.fitness.values

# Evaluar a los individuos con una condición física inválida
invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
fitnesses = map(toolbox.evaluate, invalid_ind)
for ind, fit in zip(invalid_ind, fitnesses):
    ind.fitness.values = fit

# La población es reemplazada enteramente por la descendencia.
pop[:] = offspring

# Reúna todos los ejercicios en una lista e imprima las estadísticas
fits = [ind.fitness.values[0] for ind in pop]

logbook.record(gen=g, evals=len(invalid_ind), **stats.compile(pop))
print(logbook.stream)
MIN.append(min(fits))

for i in range (g-1,g):
```

```
        if MIN [i-1]==MIN[i]:
            n=n+1
        else:
            n=0

tiempo_final = time()

tiempo_ejecucion = round(tiempo_final - tiempo_inicial, ndigits=2)

print ("")
print("-- Fin de evolución (exitosa) --")
print ("")
print ('El tiempo de ejecucion fue:',tiempo_ejecucion, 'seg') #En segundos
print ("")

best_ind = tools.selBest(pop, 1)[0]
print("Mejor individuo es %s, %s" % (best_ind, best_ind.fitness.values))

for id in range (len(cant_link)):
    r=es.ENsetlinkvalue((id+1),diametros,best_ind[id])
    f=os.path.join('RedModificadaGA.inp')
    es.ENsaveinpfile(f)

es2=EPANetSimulation(f)
es2.run()

Q2=[] #caudal transportado
V2=[] #velocidad del flujo
VA2=[] #valor
P2=[] #Presiones
```

```
    D_new=[]

    for id in range (1,len(cant_link)+1):
        D_new.append(es2.ENgetlinkvalue(id,diametros)[1])
        L.append(es2.network.links[id].results[longitudes][0])
        Q2.append(es2.network.links[id].results[caudales][0])
        V2.append(es2.network.links[id].results[velocidades][0])

a2=round((es2.ENgetlinkvalue(id,longitudes)[1])*(1.4078*(es2.ENgetlinkvalue(id,diametr
os)[1])**2.0707))
        VA2.append(a2)

    lista_links=list(es2.network.links)[:len(cant_link)+1]
    id_links=[es2.network.links[x].id for x in lista_links]

    ### Mostrar Resultados Nuevos####
    tabla_id_tuberia=pd.DataFrame(np.array(id_links).reshape(len(cant_link),1),columns= ['
Id'])
    tabla_l= pd.DataFrame(np.array(L).reshape(len(cant_link),1), columns =
['Longitud(m)'])
    tabla_d_new= pd.DataFrame(np.array(D_new).reshape(len(cant_link),1), columns =
['Diametro nuevo'])
    tabla_va2= pd.DataFrame(np.array(VA2).reshape(len(cant_link),1), columns = ['Valor
modificado'])

    tabla_diametros=pd.concat([tabla_id_tuberia,tabla_l, tabla_d,
tabla_d_new],axis=1).ffill() #concatenar
    tabla_costos=pd.concat([tabla_id_tuberia, tabla_va, tabla_va2],axis=1).ffill()
#concatenar
```

```
    print('')
    print('')

    print(tabla_diametros)
    print('')
    print(tabla_costos)

    array2 = np.array(VA2)
    salida2 = np.sum(array2)
    print('')
    print('')
    print("El costo total de la obra antes de modificar es: $","%.0f" %salida)
    print("El costo total de la obra modificada es: $","%.0f" %salida2)

    porcentaje=(((salida-salida2)/salida)*100)
    print('')
    print("El costo se redujo en: ","%.2f" %porcentaje, "%")

    plt.title("Algoritmo Genetico")
    plt.xlabel("Generacion")
    plt.ylabel("Costo de la red")
    plt.plot(MIN)
    plt.show()

if __name__ == "__main__":
    main()
```

Anexo 3. Código PSO (Particle Swarm Optimization)

```
# TESIS DE GRADO
# OPTIMIZACION DEL DISEÑO DE UNA RED DE DISTRIBUCION DE AGUA POTABLE
# PARTICLE SWARM OPTIMIZATION
# CREADO POR:  Henry Omar Peñaloza Mantilla
# ECI



#importar dependencias
import pandas as pd
import numpy as np
import os
import random
import operator
import pyfiglet
import matplotlib.pyplot as plt
import math

import EC_EVAL

from epanettools.epanettools import EPANetSimulation, Node, Link, Network, Nodes,Links, Patterns, Controls, Control # importar todos los elementos necesarios
from time import time
from deap import creator, base, tools, algorithms

pd.options.display.float_format ='{:.2f}'.format

ascii_banner = pyfiglet.figlet_format("Particle Swarm Optimization PSO")
```

```
print(ascii_banner)

#Ruta de acceso Red original
archivo = str(input("Ingrese el nombre del archivo a optimizar (incluya la extenson.inp): "))

path_Inp = archivo;

es=EPANetSimulation(path_Inp)

diametros=Link.value_type['EN_DIAMETER']
longitudes=Link.value_type['EN_LENGTH']
caudales=Link.value_type['EN_FLOW']
velocidades=Link.value_type['EN_VELOCITY']
presiones=Node.value_type['EN_PRESSURE']

cant_link=es.network.links #cantidad de tubos
cant_node=es.network.nodes #cantidad de nodos

L=[] #longitud
V=[] #velocidad del flujo
D_new=[] #nuevo diametro escogido
P_pre=[]
PEN=[]


VA=[] #valor
D=[]

for id in range (1,len(cant_link)+1):
    D.append(es.ENgetlinkvalue(id,diametros)[1])
```

```
a=round((es.ENgetlinkvalue(id,longitudes)[1])*(1.4078*(es.ENgetlinkvalue(id,diametros)[
1])**2.0707))
   VA.append(a)

lista_links=list(es.network.links)[:len(cant_link)+1]
id_links=[es.network.links[x].id for x in lista_links]
### Mostrar Resultados Iniciales####
tabla_d= pd.DataFrame(np.array(D).reshape(len(cant_link),1), columns = ['Diametro
Inicial'])
tabla_va= pd.DataFrame(np.array(VA).reshape(len(cant_link),1), columns = ['Valor
inicial'])

array = np.array(VA)
salida = np.sum(array)

DiametrosComerciales=(82.04, 105.52, 155.32, 202.17, 252.07, 298.95, 328.26, 375.16,
422.04, 468.94, 562.72)

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Particle", list, fitness=creator.FitnessMin, speed=list,
   smin=None, smax=None, best=None)


def generate(size, smin, smax):
   part = creator.Particle(random.choice(DiametrosComerciales) for _ in range(size))
   part.speed = [random.uniform(smin, smax) for _ in range(size)]
   part.smin = smin
   part.smax = smax
   return part
```

```
def updateParticle(part, best, phi1, phi2):
    u1 = (random.uniform(0, phi1) for _ in range(len(part)))
    u2 = (random.uniform(0, phi2) for _ in range(len(part)))
    v_u1 = map(operator.mul, u1, map(operator.sub, part.best, part))
    v_u2 = map(operator.mul, u2, map(operator.sub, best, part))
    part.speed = list(map(operator.add, part.speed, map(operator.add, v_u1, v_u2)))
    for i, speed in enumerate(part.speed):
        if abs(speed) < part.smin:
            part.speed[i] = math.copysign(part.smin, speed)
        elif abs(speed) > part.smax:
            part.speed[i] = math.copysign(part.smax, speed)
    part[:] = list(map(suma, part, part.speed))
    return part

DIM=len (cant_link)

def closest(K):
    lst=(82.04, 105.52, 155.32, 202.17, 252.07, 298.95, 328.26, 375.16, 422.04, 468.94,
562.72)
    r=lst[min(range(len(lst)), key = lambda i: abs(lst[i]-K))]
    return r

def suma(i, j):
    resultado=i+j
    resultado=closest(resultado)
    return resultado

def evalfit(Particle):
    return EC_EVAL.evalfit2(features=Particle,es=es)
```

```
# Structure initializers
toolbox = base.Toolbox()
toolbox.register("Particle", generate, size=DIM, smin=-50, smax=50)
toolbox.register("population", tools.initRepeat, list, toolbox.Particle)
toolbox.register("update", updateParticle, phi1=2.0, phi2=2.0)
toolbox.register("evaluate", evalfit)

def main():

    poblacion = int(input("Ingrese cantidad de particulas - poblacion inicial: "))

    pop = toolbox.population(n=poblacion)

    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", np.mean)
    stats.register("std", np.std)
    stats.register("min", np.min)
    stats.register("max", np.max)

    logbook = tools.Logbook()
    logbook.header = ["gen", "evals"] + stats.fields

    best = None
    g = 0    # Seguimiento variable del número de generaciones
    n=0  # Variable - Criterio de parada
    MIN=[]  # Lista de minimos

    tiempo_inicial = time()
```

```
while n < 100:

    g = g + 1

    for part in pop:
        part.fitness.values = toolbox.evaluate(part)
        if not part.best or part.best.fitness < part.fitness:
            part.best = creator.Particle(part)
            part.best.fitness.values = part.fitness.values
        if not best or best.fitness < part.fitness:
            best = creator.Particle(part)
            best.fitness.values = part.fitness.values
    for part in pop:
        toolbox.update(part, best)

    fits = [ind.fitness.values[0] for ind in pop]

        # Reuna todos los ejercicios en una lista e imprima las estadísticas
    logbook.record(gen=g, evals=len(pop), **stats.compile(pop))
    print(logbook.stream)

    MIN.append(min(fits))

    # Criterio de parada
    for i in range (g-1,g):
        if MIN [i-1]==MIN[i]:
            n=n+1
        else:
            n=0
```

```
tiempo_final = time()

tiempo_ejecucion = round(tiempo_final - tiempo_inicial, ndigits=2)

print('')
print ("")
print ('El tiempo de ejecucion fue:',tiempo_ejecucion, 'seg') #En segundos
print ("")
print('')
print("Mejor individuo es %s, %s" % (best, best.fitness.values))

for id in range (len(cant_link)):
    r=es.ENsetlinkvalue((id+1),diametros, best[id])
    f=os.path.join('RedModificadaPSO.inp')
    es.ENsaveinpfile(f)

es2=EPANetSimulation(f)
es2.run()

Q2=[] #caudal transportado
V2=[] #velocidad del flujo
VA2=[] #valor
P2=[] #Presiones
D_new=[]

for id in range (1,len(cant_link)+1):
    D_new.append(es2.ENgetlinkvalue(id,diametros)[1])
    L.append(es2.network.links[id].results[longitudes][0])
    Q2.append(es2.network.links[id].results[caudales][0])
    V2.append(es2.network.links[id].results[velocidades][0])
```

```
a2=round((es2.ENgetlinkvalue(id,longitudes)[1])*(1.4078*(es2.ENgetlinkvalue(id,diametros)[1])**2.0707))
        VA2.append(a2)

    lista_links=list(es2.network.links)[:len(cant_link)+1]
    id_links=[es2.network.links[x].id for x in lista_links]
    ### Mostrar Resultados Nuevos####
    tabla_id_tuberia=pd.DataFrame(np.array(id_links).reshape(len(cant_link),1),columns= ['Id'])
    tabla_l= pd.DataFrame(np.array(L).reshape(len(cant_link),1), columns = ['Longitud(m)'])
    tabla_d_new= pd.DataFrame(np.array(D_new).reshape(len(cant_link),1), columns = ['Diametro nuevo'])
    tabla_va2= pd.DataFrame(np.array(VA2).reshape(len(cant_link),1), columns = ['Valor modificado'])

    tabla_diametros=pd.concat([tabla_id_tuberia,tabla_l, tabla_d, tabla_d_new],axis=1).ffill() #concatenar
    tabla_costos=pd.concat([tabla_id_tuberia, tabla_va, tabla_va2],axis=1).ffill() #concatenar

    print('')
    print('')
    print('')
    print(tabla_diametros)
    print('')
    print(tabla_costos)

    array2 = np.array(VA2)
    salida2 = np.sum(array2)
    print('')
```

```
    print('')
    print("El costo total de la obra antes de modificar es: ","%.2f" %salida)
    print("El costo total de la obra modificada es: ","%.2f" %salida2)

    porcentaje=(((salida-salida2)/salida)*100)
    print('')
    print("El costo se redujo en: ","%.2f" %porcentaje, "%")

    plt.title("Particle Swarm Optimization")
    plt.xlabel("Generacion")
    plt.ylabel("Costo de la red")
    plt.plot(MIN)
    plt.show()

    return pop, logbook, best

if __name__ == "__main__":
    main()
```

Anexo 4. Manual de usuario

Requerimientos técnicos para el uso de los algoritmos diseñados

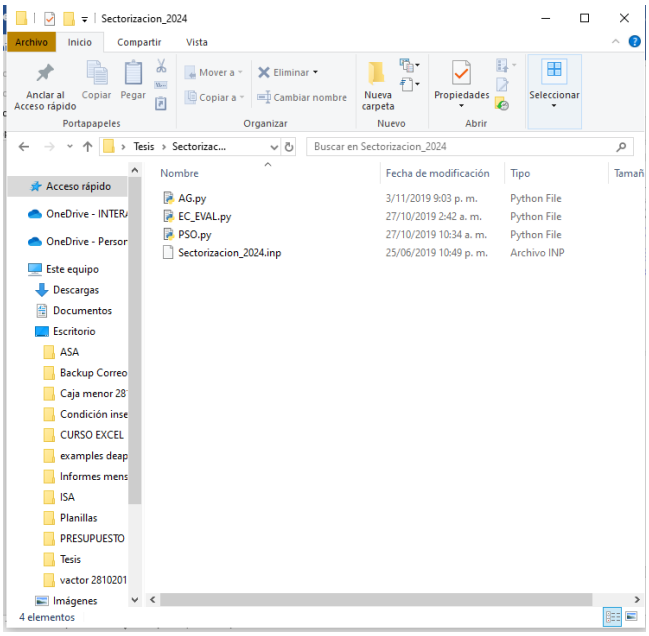
- Windows 10
- Epanet 2.0
- Python 3.7.3

Librerías de Python requeridas

- pandas
- numpy
- os
- operator
- random
- matplotlib
- pyfiglet
- math
- time
- epanettools
- deap

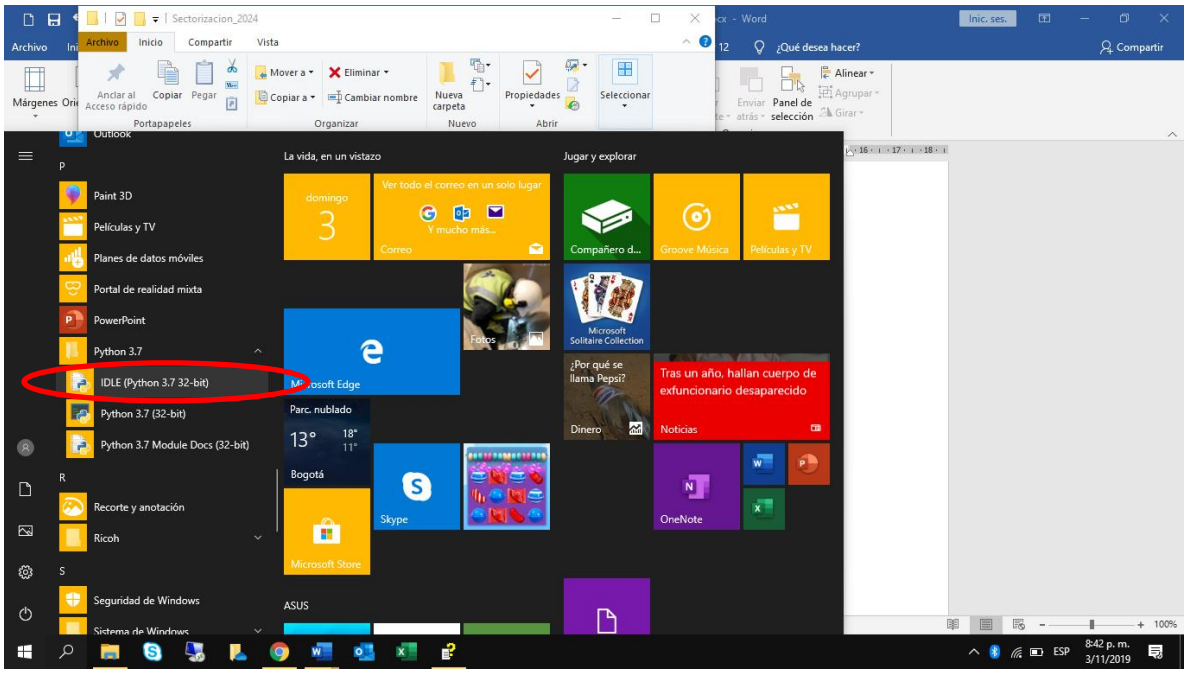
Pasos previos

1. Crear el archivo en Epanet 2.0 que describa la red que será objeto de optimización.
2. Exportar como archivo con extensión *.*inp* la red creada en Epanet 2.0.
3. Crear una carpeta e incluir en esta los archivos *AG.py*, *PSO.py* y *EC_EVAL.py* objeto de este manual, así como el archivo con extensión *.*inp* que contiene la red creada en Epanet 2.0.

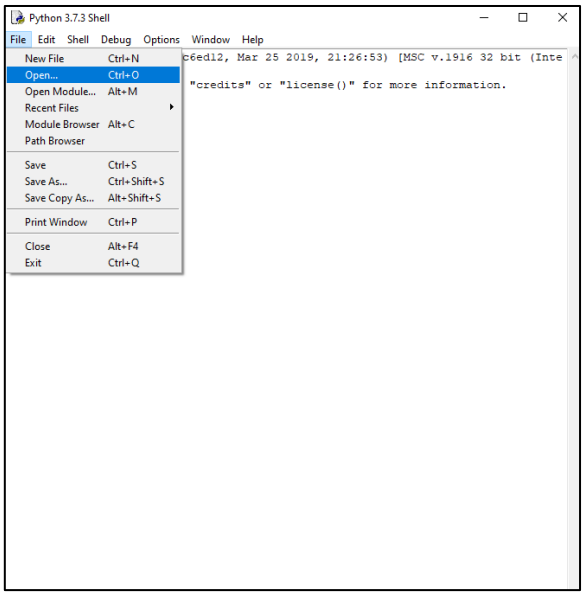


Acceso al Sistema

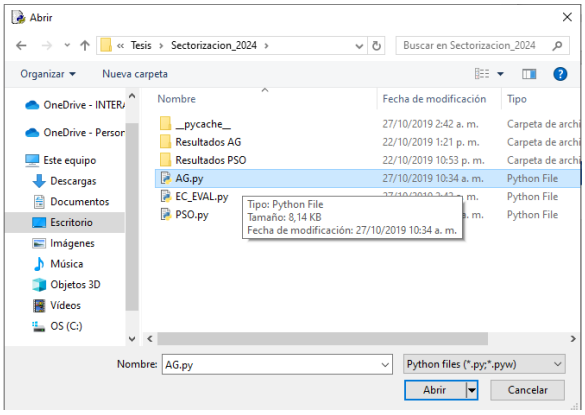
1. Desde el menú *Inicio* haga click en la carpeta: *Python 3.7*.
2. Posteriormente, haga click sobre el icono: *IDLE (Python 3.7 32-bit)*, el cual abre la ventana: *Python 3.7.3 Shell*.



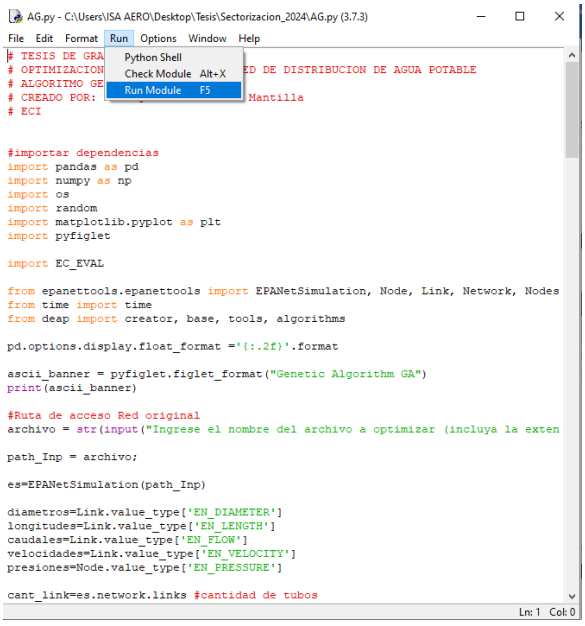
3. Desde el menú *File* haga click en la opción *Open*... la cual desplegará la ventana *Abrir*.



4. Desde la ventana Abrir seleccione el Código *AG.py* o *PSO.py* según sea el método de optimización que desee aplicar.

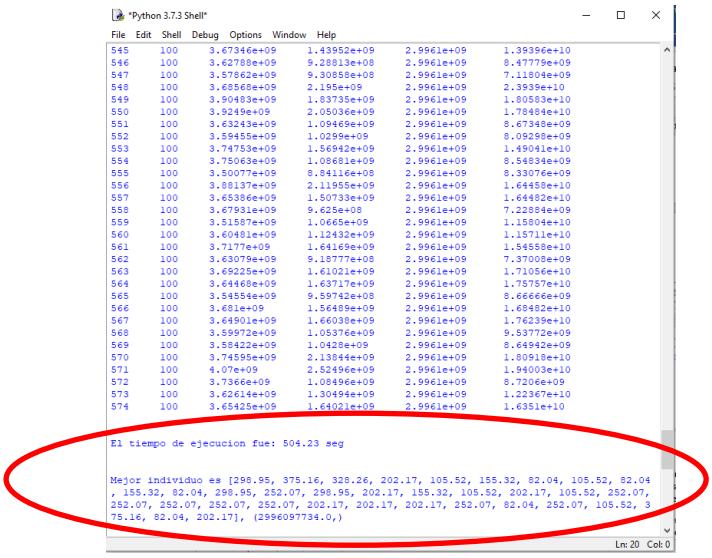


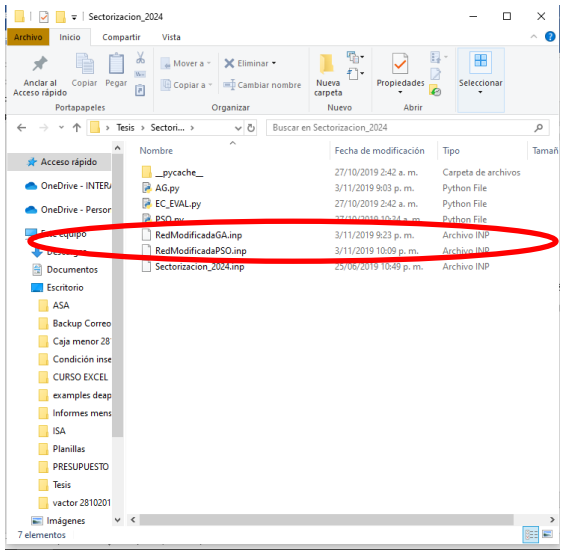
5. Una vez se encuentre en el Código seleccionado despliegue el menú *Run* y haga click en *Run Module*.



6. Para iniciar la ejecución el módulo seleccionado, ingrese los datos requeridos oprimiendo *Enter* después de cada dato ingresado.

Al culminar la optimización se obtienen un listado con los nuevos diámetros, el valor total de la red optimizada y el tiempo utilizado para su ejecución. Adicionalmente, se genera en la misma carpeta un nuevo archivo con extensión **.inp* que contiene la red optimizada lista para ser simulada en Epanet (*RedModificadaAG* o RedModificadaPSO según haya sido el método de optimización escogido).





Modificaciones al Sistema

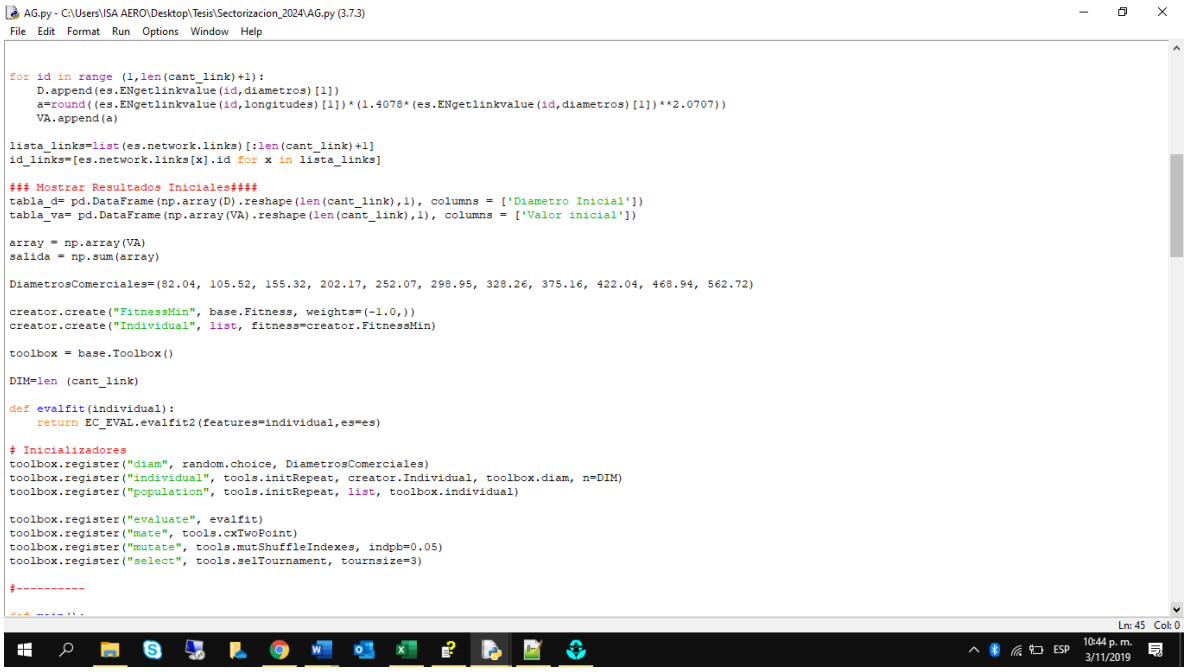
Por último, a continuación se describen los pasos que deben seguirse en caso que requiera modificar los diámetros comerciales que va a tener en cuenta en el proceso de optimización y/o las restricciones hidráulicas aplicables.

- Para modificar los diámetros comerciales es necesario:

  Modificar en el Código *AG.py*:

  La ecuación de costos en las líneas 52 y 209.

  Los diámetros en la línea 65 (incluirlos en mm).



  Modificar en el Código *PSO.py*:

  La ecuación de costos en las líneas 57 y 201.

  Los diámetros en las líneas 69 y 100 (incluirlos en mm).

  smin y smax en la línea 114 (tener en cuenta que se recomienda que estos valores correspondan a la diferencia entre 2 diámetros comerciales en mm).

PSO.py - C:\Users\ISA AERO\Desktop\Tesis\Sectorizacion_2024\PSO.py (3.7.3)

File Edit Format Run Options Window Help

```
caudales=Link.value_type['EN_FLOW']
velocidades=Link.value_type['EN_VELOCITY']
presiones=Node.value_type['EN_PRESSURE']

cant_link=es.network.links #cantidad de tubos
cant_node=es.network.nodes #cantidad de nodos

L=[] #longitud
V=[] #velocidad del flujo
D_new=[] #nuevo diametro escogido
P_pre=[]
PEN=[]


VA=[] #valor
D=[]

for id in range (1,len(cant_link)+1):
    D.append(es.ENgetlinkvalue(id,diametros)[1])
    a=round((es.ENgetlinkvalue(id,longitudes)[1])*(1.4078*(es.ENgetlinkvalue(id,diametros)[1])**2.0707))
    VA.append(a)

lista_links=list(es.network.links)[:len(cant_link)+1]
id_links=[es.network.links[x].id for x in lista_links]
### Mostrar Resultados Iniciales####
tabla_d= pd.DataFrame(np.array(D).reshape(len(cant_link),1), columns = ['Diametro Inicial'])
tabla_va= pd.DataFrame(np.array(VA).reshape(len(cant_link),1), columns = ['Valor inicial'])

array = np.array(VA)
salida = np.sum(array)

DiametrosComerciales=(82.04, 105.52, 155.32, 202.17, 252.07, 298.95, 328.26, 375.16, 422.04, 468.94, 562.72)

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Particle", list, fitness=creator.FitnessMin, speed=list,
    smin=None, smax=None, best=None)


def generate(size, smin, smax):
    part = creator.Particle(random.choice(DiametrosComerciales) for _ in range(size))
    part.speed = [random.uniform(smin, smax) for _ in range(size)]
```

domingo, 3 de noviembre de 2019

10:31 p. m. 3/11/2019

Modificar en el Código EC_EVAL.py:

La ecuación de costos en la línea 33.

EC_EVAL.py - C:\Users\ISA AERO\Desktop\Tesis\Sectorizacion_2024\EC_EVAL.py (3.7.3)

File Edit Format Run Options Window Help

```
    presiones=Node.value_type['EN_PRESSURE']

    cant_link=es.network.links #cantidad de tubos
    cant_node=es.network.nodes #cantidad de nodos

    VA=[]
    PEN1=[]
    PEN2=[]

    for id in range (len(cant_link)):
        r=(es.ENsetlinkvalue((id+1),diametros,features[id]))
        f=os.path.join('parcial_modificada.inp')
        a=((es.ENgetlinkvalue(id+1,longitudes)[1])*(1.4078*(es.ENgetlinkvalue(id+1,diametros)[1])**2.0707))
        VA.append(a)
    cost = np.sum(VA)

    es.ENsolveH();

    for id in range (1,len(cant_node)):
        if (es.ENgetnodevalue(id,presiones)[1]) < 20:
            pen_pre = (20-(es.ENgetnodevalue(id,presiones)[1]))*100000000
        elif (es.ENgetnodevalue(id,presiones)[1]) > 50:
            pen_pre = ((es.ENgetnodevalue(id,presiones)[1])-50)*100000000
        else:
            pen_pre = 0
        PEN1.append(pen_pre)
    PP=np.sum(PEN1)

    for id in range (1,len(cant_link)):
        if (es.ENgetlinkvalue(id,velocidades)[1]) < 0.5:
            pen_vel = (0.5-(es.ENgetlinkvalue(id,velocidades)[1]))*100000000
        elif (es.ENgetlinkvalue(id,velocidades)[1]) > 2.5:
            pen_vel = ((es.ENgetlinkvalue(id,velocidades)[1])-2.5)*100000000
        else:
            pen_vel = 0
        PEN2.append(pen_vel)
    PV=np.sum(PEN2)

    costo=round(cost+PP+PV)

    return costo,
```

Ln: 62 Col: 36

10:47 p. m. 3/11/2019

- Para modificar las restricciones impuestas al diseño:

Modificar en el Código *EC_EVAL.py*:

Presión mínima y máxima, líneas 40 a 43.

Velocidad mínima y máxima, líneas 50 a 53.