FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

CENTRO DE INVESTIGACIÓN EN INGENIERÍA Y CIENCIAS APLICADAS.

Algoritmo Evolutivo en Ambiente GRID para el Problema de Diseño de Redes de Distribución de Agua

TESIS PROFESIONAL

PARA OBTENER EL GRADO DE:

DOCTORA EN INGENIERÍA Y CIENCIAS APLICADAS,
CON OPCIÓN TERMINAL EN TECNOLOGÍA ELÉCTRICA

P R E S E N T A:

MC. Erika Yesenia Ávila Melgar

Asesor: Dr. Marco Antonio Cruz Chávez

LABORATORIO DE OPTIMIZACIÓN Y SOFTWARE

Cuernavaca, Morelos. **Marzo de 2015**

Es fácil proponer soluciones complejas para problemas sencillos, pero algo realmente complicado es proponer soluciones sencillas para resolver problemas complejos (anónimo)

Para entender el problema es importante escuchar; para resolver el problema es necesario sentir (anónimo)

DEDICATORIAS

A Dios:

Por permitirme estar aquí.

A mis Padres:

Por permitirme el milagro de tener vida

A mis Hermanos:

Por estar siempre presentes en mi vida

A mis Sobrinos:

Por ser siempre alegría.

A los lectores de este trabajo:

Este trabajo fue realizado de inicio a fin pensando en ellos.

A mi director de tesis:

Por apoyarme siempre de inicio a fin

A Marco:

Por llegar a mi vida y ser importante en ella.

A André:

Por llegar a mi vida y enseñarme lecciones importantes que en la escuela no había aprendido; por ser mi mayor fuente de inspiración y lucha; por permitirme conocer la expresión real del amor.

AGRADECIMIENTOS

En este espacio quisiera expresar mi gratitud a todos quienes han contribuido de manera directa o indirecta en la realización de este trabajo de tesis doctoral. Sin embargo, enumerar de manera explícita a todos será casi imposible porque la lista es interminable…

De manera especial quiero agradecer, por la colaboración económica:

A **CONACYT** por brindarme los recursos económicos que me hicieron sentir privilegiada por contar con la beca, mes con mes durante tres años, recordándome cada vez que la recibía mi gran compromiso con la sociedad. Sin duda la beca CONACYT me motivó a dar lo mejor de mí en este trabajo doctoral.

A la **Universidad Autónoma del Estado de Morelos**, y en especial al **Centro de Investigación en Ingeniería y Ciencias Aplicadas**, por permitirme formar parte de la institución y del grupo respectivamente. Gracias por otorgarme las herramientas necesarias para mi trabajo doctoral. Gracias también por el apoyo económico para la publicación de artículos y para la estancia en la universidad de Almería.

A la **fundación BBVA Bancomer** por la beca de movilidad que me permitió solventar mejor los gastos en un país extraño y lejano: España.

A la **fundación Telmex** por darme el privilegio de pertenecer a los becarios Telmex, facilitándome además la beca y el equipo de cómputo que indudablemente fueron de gran ayuda para el desarrollo del proyecto doctoral.

A los organizadores del evento "Second EELA-2 Conference", por otorgarme los recursos económicos que me permitieron asistir al congreso, en Choroni Venezuela en Noviembre de 2009, para representar al grupo con la presentación del artículo.

También quiero expresar mi más sincero agradecimiento a todas las personas que me apoyaron en los aspectos académicos, moral y trámites administrativos para la realización de este trabajo. En especial quiero agradecer de forma explícita:

Al director del CIICAp: Dr. Pedro Márquez por todo el apoyo desde el inicio de mi estancia en el CIICAp.

A mi director de tesis: Dr. Marco Antonio Cruz Chávez, agradezco enormemente su confianza en mí y en mí proyecto. Agradezco mucho todo su tiempo, su apoyo, sus comentarios y sus observaciones acertadas para el desarrollo de este trabajo doctoral.

A mis revisores: nuevamente al **Dr. Marco Antonio Cruz Chávez**, al **Dr. David Juárez Romero**, al **Dr. Abelardo Rodríguez León**, a la **Dra. Margarita Tecpoyotl Torres**, al **Dr. Álvaro Zamudio Lara**, al **Dr. Martín Martínez Rangel** y al **Dr. Martín Heriberto Cruz Rosales**. Muchas gracias a todos por su tiempo y trabajo empleado en la revisión de este proyecto de tesis; gracias también por sus comentarios acertados que me permitieron hacer de éste un mejor trabajo.

Al personal de la Universidad Autónoma del Estado de Morelos por su apoyo, por su amabilidad y sobre todo por realizar los trámites administrativos de inicio a fin. Gracias también por la gestión de las becas CONACYT y de movilidad.
Gracias también al consejo técnico, a servicios escolares y a todos quienes contribuyeron en la publicación de las convocatorias extemporáneas para titulación. Después de dos años de larga espera este proceso me permitió obtener mi certificado de doctorado.

A todos mis profesores. Con admiración, respeto, y mucho cariño quiero agradecerles por transmitirme sus conocimientos, por enseñarme grandes lecciones que me han permitido formarme en el ámbito educativo. Gracias a todos mis profesores tanto de educación básica como de educación superior.

Gracias a a todos mis queridos profesores de educación básica. En especial gracias a los profesores: Mercedes Díaz Flores y Pedro Melgar Acosta.

Gracias a mis queridos profesores, y ahora compañeros de trabajo de la **Preparatoria Abierta de la Universidad Autónoma del Estado de Morelos** quienes, sin duda, contribuyeron y continúan contribuyendo en mi formación, siendo ahora parte de un excelente ambiente de trabajo. En especial, muchas gracias a los profesores: Bióloga. Eva Judith Godínez López, Bióloga. Beatriz Mejía Cícero, Biólogo. Salvador Santillán y Lic. María Eugenia. Gracias por su confianza, por su apoyo y por su amistad.

Gracias a todos mis queridos profesores del **Instituto Tecnológico de Zacatepec**, en especial muchas gracias a los profesores: M.T.I María Isabel Vásquez Ocampo, Ing. Alma Delia Moreno Altamirano, Ing. Andrés Hernández Peralta, Lic. Alejandro Morales Lizama, Dr. Héctor Medellín Hernández, Ing. Jesús Salvador Torres Peralta, M.C. Mario Humberto Tiburcio Zuñiga y demás profesores, cuyos rostros recuerdo perfecto por todas sus enseñanzas.

Gracias a todos mis queridos profesores del **Centro Nacional de Investigación y Desarrollo Tecnológico**, en especial muchas gracias a los profesores: Dr. Víctor Jesús Sosa, Dr. René Santaolalla, Dr. Máximo López.

Gracias a todos mis queridos profesores del **Centro Nacional de Investigación en Ingeniería y Ciencias Aplicadas**, en especial muchas gracias a los profesores: Dr. Marco Antonio Cruz Chávez, Dr. David Juárez Romero, Dr. Crispín Zavala Díaz, Dr. Abelardo Rodríguez León y Dr. Carlos Eduardo Mariano Romero.

Muchas Gracias a todas las Instituciones Académicas de las cuales fue un honor formar parte, y siempre recordaré con mucha gratitud y cariño: **Primaria:** Resurgimiento, Apancingo Morelos; **Secundaria:** Tlacaelel y Celia Muñoz Escobar, Coatlan del Río Morelos y Apancingo Morelos, respectivamente; **Preparatoria:** Preparatoria Abierta del Estado de Morelos, de la Universidad Autónoma del Estado de Morelos, Cuernavaca Morelos; **Licenciatura:** Instituto Tecnológico de Zacatepec, Zacatepec Morelos; **Maestría:** Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca Morelos; **Doctorado:** Centro Nacional de Investigación en Ingeniería y Ciencias Aplicadas, de la

Facultad de Ciencias Químicas e Ingeniería en la Universidad Autónoma del Estado de Morelos, Cuernavaca Morelos.
Gracias también al Centro de Estudios de Lenguas Extranjeras de la Universidad Autónoma del Estado de Morelos.

Muchas gracias también a todos los investigadores del mundo entero que han compartido sus conocimientos a través de publicaciones y libros. Gracias por apoyarme con conocimientos de diversos temas (Redes de Distribución de Agua, Algoritmos Evolutivos, Computo Paralelo, Sistemas Distribuidos, Ambientes Grid, entre otros); todos muy útiles para lograr el objetivo planteado en esta tesis doctoral.

Muchas Gracias a los organizadores del evento “Second EELA-2 Conference”, quienes aceptaron un artículo llamado “Parallel Hybrid Evolutionary Algorithm in a Grid Environment for the Job Shop Scheduling Problem” del grupo de Optimización y Software, al cual pertenezco. Muchas gracias también a los organizadores por otorgarme los recursos económicos que me permitieron asistir al congreso, en Choroni Venezuela en Noviembre de 2009, para representar al grupo con la presentación del artículo. Este congreso, sin duda, además de ampliar conocimientos en varios temas también me permitió conocer investigadores de diferentes países y tener lindas amistades, entre otras: Indira, Raquel Pezoa y Alina Roig.

A MIS QUERIDOS Y GRANDES AMIGOS DE TODA LA VIDA…

A todos mis amigos y compañeros de la infancia, en especial a: mi adorada hermanita Yuri. Gracias también a mis amigos: Normita y Lety; Piedad; Sonia y Geo; Sai, Homero y Oscar, Fabi, Eliza y Edgar; Elizabeth y Ricardo.

A todos mis amigos y compañeros de primaria, en especial a: Nally y Sandra, presentes en mí con hermosos recuerdos. **A todos mis amigos y compañeros de secundaria, en especial a:** Karla, Xochi, Leni y Nally.

A todos mis amigos y compañeros de preparatoria, en especial a: Julio y Ana María.

A mis amigos que conocí en la época del Estilismo y con quienes he compartido muy gratos momentos: Elenia Villalobos, Laura Estrada, Laura Suarez, Marco Ramírez, Marco Miranda, Maricruz Botello y Leticia Díaz.

A todos mis amigos y compañeros de licenciatura, en especial a: Blanca, Miriam, Teo, Samuel, Albert, Vianey, Yuli, Richard, Marco Polo, Víctor Salgado, Jesús Melgar, Alejandro Figueroa, Alma Cervantes, Leidi, Mariana, Citlali, Bety, Laura, Irani, Vladimir, Hilda y Karime… Donde quiera que se encuentren “Muchas Gracias por su linda amistad”.

A todos mis amigos y compañeros de Maestría, en especial a: Marco, Mireya, Vero, Ariel, Tito y Armando. Mil gracias también al grupo de sistemas distribuidos de quienes conservo bellos recuerdos… Gracias al Dr. Víctor Sosa y al Dr. René Santaolalla. Gracias también a mis amigos Lupita, Selene, Rosario, Maricela, Roberto, Gustavo y Sheila.

A todos mis amigos y compañeros de Doctorado, en especial a: Mireya, **Alina**, y **Pedro**, por su amistad incondicional, por su comprensión, por demostrarme su afecto con apoyo constante y palabras de aliento en los momentos difíciles… y principalmente millones de gracias por todos los momentos tan agradables de convivencia, en los que la riso-terapia estuvo siempre presente…

Gracias también a mis amigas: **Ocotlán Díaz Parra, Beatriz Martínez y Yesica Calderón,** por todo su apoyo, amistad y por confiar en mí**.**

También agradezco el apoyo de todos mis compañeros del CIICAp, en especial agradezco a Fredy por ser parte importante del equipo de trabajo y por compartir conocimientos logrando beneficio mutuo.

A la Universidad de Almería que me abrió sus puertas para realizar la estancia doctoral en el periodo Enero-Abril de 2010.

Al Dr. Raúl Baños, por aceptarme en la estancia doctoral, por regalarme su valioso tiempo y por compartirme conocimientos invaluables sobre el Diseño de Redes de Distribución de Agua y Algoritmos Evolutivos. Gracias por su amistad, por sus consejos, por su apoyo, por su tiempo y por sus acertados comentarios.

A la familia Gil Montoya, personas encantadoras. En especial muchas gracias: **a la Dra. Consolación Gil y a la Dra. María Dolores Gil,** por todas sus atenciones y por compartirme conocimientos invaluables de diversas áreas, muchas gracias también por abrirme las puertas de su casa, por todo su apoyo y por su linda amistad.

A todos los compañeros y amigos de la Universidad de Almería, en especial al Dr. Juan Reca, Antonio López Márquez, Antonio Fernández Molina, Ramón González Sánchez y Laura Da Silva. Gracias por aceptarme en "el despacho". Gracias por su apoyo y por su amistad.

A la familia Sánchez del Pino, porque fueron muy lindas personas conmigo, durante toda mi estancia en Almería, realmente generosos y amables. Gracias por todo el apoyo, por todo el cariño y por todas sus atenciones.

A Jessica Analía Gómez, Elena Martínez y Elena Tipaz, grandes amigas y excelentes compañeras de paseos en Almería y Madrid.

A la familia Brown Lee por abrirnos recibirnos en su casa. Por apoyarnos siempre. Por su hermosa amistad. Por la confianza. Sobre todo, muchas gracias por permitirnos tener, en Newport, las vacaciones que jamás imaginé.

A Marco y a André, por ser mi familia, por estar siempre cerca, por toda su paciencia, por todo el cariño y por todo su apoyo en este gran reto, por compartir conmigo experiencias únicas, irrepetibles e inolvidables…

A mis Padres, porque sin escatimar esfuerzo alguno, me han dado siempre confianza, apoyo libertad y sobre todo una familia maravillosa. Gracias por darme la vida. Gracias por su tiempo, por su apoyo, por su comprensión y por los paseos tan divertidos…
Con admiración, respeto y gran amor:
A mi padre, le agradezco mucho por ser un ejemplo de bondad, lucha y fortaleza.
A mi madre, le agradezco por creer en mí, por darme tanto apoyo y amor.
A todos mis hermanos (en especial a mis hermanas), les agradezco por estar presentes en todo momento. Gracias por confiar en mí y por apoyarme tanto, motivándome a alcanzar mis metas; gracias por ser las personas que más han cuidado de mí y de mi hijo.
Gracias también a todos mis sobrinos, a quienes quiero tanto, con mención especial para Ari, Pepe, Isra, Eduardo, Sebastián, Andrea, por ser para André como verdaderos hermanos… siempre cuidándolo y apoyándolo.
Gracias también a Aldo y a Eliza por ser los hermanos pequeños de André.
Gracias a Nor y Belza por ser para André amor, juego y diversión.
Gracias a Cristian y a Tlalli que me han apoyado mucho.
Gracias también a mi tía Cire y Yola, lindísimas personas que me han recibido en su casa, siendo parte de las vacaciones tan bonitas.
Gracias también a mis queridos padrinos: Estela y Félix; Mercedes y Fernando; Estela Melgar y Guadalupe Ávila.

Gracias a todos y cada uno de los integrantes de la familia, que por cierto son cientos pero todos muy queridos: Abuelos, Padres, Padrinos, Suegros, Cuñados, Tíos, Primos; cada uno con un lugar especial en mí.

Gracias a mis queridos alumnos que se esfuerzan y aprenden…

Gracias a los lectores de este trabajo.

Gracias a los autores de algunos de mis libros favoritos que me ayudaron a entender mejor mis emociones y mantenerme de pie emocionalmente. En especial gracias a: Eckart Tolle, Katie Byron, Osho, John Kabat Zinn, entre muchos otros.

Gracias también a toda la humanidad, pero en especial a aquellas personas que cumplen su misión en este mundo con alegría y entusiasmo. Aquellas personas solidarias con la naturaleza y con los demás seres vivos. Personas que dan todo sin esperar recibir algo a cambio; que dan siempre lo mejor de sí mismas solo para ayudar; todas estas personas tienen mi admiración, mi respeto y mi más profundo agradecimiento. Y muchas de las personas que no me fue posible mencionar, se encuentran en esta lista.
MUCHAS GRACIAS.

Finalmente, es muy importante para mí dar gracias a **Dios** por permitirme estar aquí. Por permitirme disfrutar de la naturaleza, de sus elementos, de mi familia. Dar gracias también porque a pesar de tantos obstáculos y contratiempos siempre me dio coraje, fortaleza, paciencia, sabiduría, fuerza de voluntad, fe y perseverancia; todos los elementos necesarios para levantarme cada día y luchar hasta alcanzar mi más grande meta en el ámbito profesional: “**concluir este trabajo doctoral lo mejor posible”**.

Resumen

Actualmente existen problemas complejos de optimización, tanto en la comunidad científica como en la sociedad. Estos problemas deben ser resueltos encontrando siempre la mejor solución para el problema. Gracias a los avances tecnológicos es posible diseñar y ejecutar algoritmos eficientes que encuentran soluciones adecuadas a diferentes problemas de optimización combinatoria en tiempos de cómputo limitados. Sin embargo, para problemas reales, de grandes dimensiones, encontrar la mejor solución en un tiempo relativamente corto no es una tarea trivial.

El diseño de algoritmos, hace algunas décadas, se enfocaba en el uso de métodos de solución exactos que garantizaban encontrar la mejor solución de un problema, pero en la mayoría de los casos se trataba de problemas teóricos que podían ser resueltos en tiempos de cómputo limitados. En la actualidad, encontrar la mejor solución a los problemas reales utilizando métodos de solución exactos conlleva altos costos temporales y computacionales. En estos casos, los métodos de solución heurísticos representan una mejor alternativa. Para problemas de optimización reales de gran envergadura, los métodos heurísticos prometen encontrar soluciones de calidad en un tiempo razonable.

La implementación de métodos heurísticos requiere también la existencia de recursos temporales y espaciales, ya que estos métodos incluyen diversos parámetros que deben ajustarse, mediante una serie de experimentos, para obtener así el mejor comportamiento del algoritmo cuando soluciona problemas. Sin embargo, una vez que se encuentran los valores paramétricos con los que el algoritmo puede resolver el problema, los métodos heurísticos pueden resolver problemas de optimización en tiempos de cómputo limitados.

En la actualidad una excelente alternativa para abordar problemas reales, es utilizar estrategias heurísticas en combinación con estrategias paralelas que, generalmente al trabajar en conjunto, permiten minimizar el tiempo requerido para encontrar la mejor solución a un problema. Así, el uso de plataformas que habilitan el cómputo paralelo, como los clúster de computadoras y las plataformas grid, han sido y son imprescindibles y de gran importancia en la búsqueda de la mejor solución para problemas reales. La grid ofrece gran cantidad de recursos computacionales que facilitan la ejecución eficiente de algoritmos. En este trabajo, la grid es de gran ayuda para explorar de manera exhaustiva los distintos componentes del algoritmo evolutivo paralelo. Con el algoritmo evolutivo paralelo, se reduce notablemente el tiempo de cómputo requerido (comparado con el tiempo que requiere para su ejecución el AE secuencial) para encontrar la mejor solución del problema abordado y a la vez que se encuentran soluciones de calidad para el problema de Diseño de Redes de Distribución de Agua, el cual es el contexto de este trabajo doctoral.

El problema de Diseño de Redes de Distribución de Agua consiste en encontrar la configuración de la red de costo mínimo que además satisface las necesidades hídricas de los usuarios de la red. Por su complejidad, éste es un problema de optimización combinatoria clasificado en la categoría de los problemas NP-Completos. Este problema es de gran importancia tanto para la comunidad científica como para la sociedad por su amplia aplicación práctica y por la escasez de los recursos hídricos. Este problema ha sido estudiado durante más de tres décadas, por la comunidad científica, la cual se ha enfocado principalmente en la solución de problemas teóricos. A través del tiempo, el problema WDND ha evolucionado con diferentes formulaciones matemáticas en las que difieren, las topologías de las redes, las variables de decisión e incluso los métodos de solución. Para este problema se han propuesto diferentes métodos de solución que van desde algoritmos exactos hasta algoritmos heurísticos, métodos secuenciales y métodos paralelos. Sin embargo, después de realizar una revisión detallada en la literatura no se encontró algún algoritmo evolutivo en ambiente grid para el problema abordado, por lo cual se considera un área de oportunidad para proponer un novedoso algoritmo evolutivo para dar solución al problema.

En este trabajo, se hizo un estudio exhaustivo de los componentes y operadores del algoritmo evolutivo con la finalidad diseñar un algoritmo eficiente que encontrara soluciones de calidad para problemas de Diseño de Redes de Distribución de Agua. Se realizaron una gran cantidad de experimentos para trabajar con poblaciones de diferente tamaño, trabajar con individuos factibles e infactibles. Así mismo, se hicieron experimentos para encontrar los parámetros del algoritmo, con los cuales se obtiene un mejor comportamiento en la solución del problema. También, se hicieron estudios experimentales para observar el comportamiento del algoritmo cuando se trabaja con diferentes métodos de selección, cruce y mutación. Todos estos estudios experimentales fueron posibles gracias a un modelo de paralelización propio implementado como resultado de combinar dos modelos de paralelización muy conocidos en la literatura, el modelo maestro/esclavos y el modelo de islas, de los cuales se toman las ventajas, y se evitan las desventajas, para proponer un nuevo modelo robusto conocido, en este trabajo, como modelo maestro-alumnos.

El algoritmo evolutivo en ambiente grid, después de realizar múltiples estudios experimentales, finalmente demostró encontrar soluciones de calidad en tiempos de cómputo razonables para resolver el problema de Diseño de Redes de Distribución de Agua.

Abstract

The design of efficient algorithms which can be applied to find the best solution to solve complex problems is the main goal of combinatory optimization. Currently there are complex optimization problems in the scientific community and the society. These problems must be solved by efficient algorithms in a limited computational time. So, computational space resources (how much memory it takes to solve the problem) and reasonable time (how many steps it takes to solve the problem) must be considered when solving real-world complex optimization problems.

Nowadays, thanks to the technological development advances in algorithms and computer hardware it is possible to solve real-life optimization problems that years ago were thought to be unsolvable, principally because the techniques of solving problems were based on deterministic algorithms. Currently heuristic methods represent a better alternative. Even when they use many parameters which must be adjusted, they are promising and widely used methods to solve real-world complex optimization problems, in a reasonable period of time. The results and the efficiency of heuristic methods improve significantly when they are implemented using parallel strategies. So, the cluster and the grid computing platforms have been very important for resolution of optimization problems.

The cluster and grid platforms provide many computational resources to help researchers to develop efficient computational algorithms. In this work the grid is primordial due to it helps to carry out a large number of experiments to adjust the parameters of the algorithm. Parameters of the algorithm are adjusted in order to find the best solution for the Water Distribution Network Design problem, which is the context for this thesis.

The Water Distribution Network Design, according to the computational complexity theory, is classified as NP-Complete problem. It has been widely studied, during three decades, by many researchers because of its practical applicability. The problem consists of finding the most efficient way to supply water to consumers, with given constraints, for example pressure requirements must be reached to offer users an adequate service by satisfying their water requirements.

In two first decades the optimal design of water distribution networks was based on lineal programming models, the decision variables were based on continuous variables and the solution method for the problem was principally based on lineal programming methods. The topology of the network design tended to be a branched layout.

In last decade, the Water Distribution Network Design problem has gradually been modified; it has been formulated as a non-linear programming problem and pipe diameters have been stated as discrete decision variables. The solution method for the problem has been generally based on heuristics methods like Evolutionary Algorithms (EAs), Simulated Annealing and, so on. The design of the network has been a looped layout and, the network technique to supply water to consumers has been gravity. Even though, when the problem in three decades refers to Water Distribution Network Design problem, there are some important differences, between first two decades and last decade, that make the problem slightly different: mathematical formulation, decision variables, topology, solution method, and, technique to feed the network (pumping or gravity). Even when a large variety of techniques have been proposed by many researchers, a totally satisfactory method is not available yet. Many of the proposed algorithms have been tested their performance using small or medium-sized benchmarks and they rarely have been tested using large-scale real-life networks. Moreover, the most of works focus in design of new networks and do not consider the existent networks which work inappropriately to be partially designed. In that way the Evolutionary Algorithm in grid environment, here implemented, represents a novel method for large-scale real-life Water Distribution Network Design Problem. In order to solver the Water Distribution Network Design Problem an Evolutionary Algorithm in Grid Environment has been developed and implemented in this work.

Additionally, several experimental tests with the components of the Evolutionary Algorithm have been carried out, with the aim of achieving the goal of finding quality solutions for the problem. Those experiments included space search, characteristics and size of initial populations to help the evolutionary algorithm to converge earlier to better solution. Also experiments to adjust the diverse numbers of parameters the algorithm such as number of generations, probabilities of mutation and crossover operators, were performed. All these experiments were possible by implementing a proposed parallelization model called master-students, which combines islands and master-slaves models to take advantage of both models benefits. The resultant parallel evolutionary algorithm in grid environment, here proposed and developed, obtains better results than those reported in the literature for the Water Distribution Network Design Problem.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Nomenclatura

WDND	Siglas en inglés de Diseño de Redes de Distribución de Agua, *Water Distribution Network Design.*
RDA	Redes de Distribución de Agua.
AG	Siglas en inglés de Algoritmo Genético.
EA	Siglas en inglés de Algoritmo Evolutivo, *Evolutionary Algorithm.*
SEA	Siglas en inglés de Algoritmo Evolutivo Secuencial, *Sequential Evolutionary Algorithm*.
PEA	Siglas en inglés de Algoritmo Evolutivo Paralelo, *Parallel Evolutionary Algorithm*.
SEA-WDND	Siglas utilizadas para hacer referencia al algoritmo evolutivo secuencial para el problema de Diseño de Redes de Distribución de Agua.
PEA-WDND	Siglas utilizadas para hacer referencia al Algoritmo Evolutivo Paralelo para el problema de Diseño de Redes de Distribución de Agua.
MPI	Siglas en inglés, *Message Passing Interface.* Biblioteca de funciones para cómputo paralelo que utiliza el paso de mensajes.
MPICH	Versión ampliamente usada de la librería de paso de mensajes MPI.
OpenMPI	Especificación que define directivas de compilación, bibliotecas de funciones y variables de entorno que pueden ser utilizadas para paralelismo de memoria compartida, en programas en lenguaje Fortran y lenguaje C/C++ sobre gran cantidad de plataformas.

Notación Utilizada

Red de Distribución de Agua

p	Planeación o configuración del diseño de la red.
$C(p)$	Costo del diseño de la red.
T	Conjunto de tuberías de la red.
n_d	Número de diámetros de la red.
D	Conjunto de diámetros comerciales disponibles.
n_t	Número de tuberías de la red.
N	Conjunto de nodos de la red.
n_n	Número de nodos de la red.
c_{di}	Costo de una tubería de diámetro específico.
L_{ij}	Longitud de una tubería de la red.
H_i	Presión de cabecera requerida en un nodo.
H_{min}	Presión mínima requerida.
H_{max}	Presión máxima requerida.
V_j	Velocidad en una tubería.
V_{min}	Velocidad mínima en un nodo.
V_{max}	Velocidad máxima en un nodo.
M	Mallas o circuitos de una red.

Representación de Grafos

G	Grafo.
E	Conjunto de arcos de un grafo.
$\|E\|$	Número de arcos en un grafo.
V	Conjunto de vértices de un grafo.
$\|V\|$	Número de vértices en un grafo.
v_i	Vértice origen.
v_j	Vértice destino.

Algoritmo Evolutivo Secuencial

s	Individuo o solución.
TamPob ó TP	Tamaño de población.
PrCr ó P_c	Probabilidad de cruce.
PrMt ó P_m	Probabilidad de mutación.
OpMt ó O_m	Operador de mutación.
n_g ó NG	Número de generaciones.
n_e	Número de ejecuciones del algoritmo.
C_{min}	Mejor aptitud de un individuo, representada con el costo mínimo del diseño de la red.

Modelo del Algoritmo Evolutivo Paralelo

GR	Conjunto de grupos del algoritmo paralelo.
|GR| ó NGr	Número de grupos en los que se ejecuta el algoritmo.
TamGr	Número elementos de un grupo.
Gr	Grupo compuesto por un maestro y alumnos.
P	Conjunto de procesos que fungen como maestros o profesores.
np	Número de procesos.
|P|	Número de procesos maestros o profesores.
A	Conjunto de procesos que fungen como esclavos o alumnos.
|A| ó NumAlum	Número de procesos esclavos o alumnos.
r ó T_e	Tiempo de entrega de soluciones, conocida como frecuencia de migración en el modelo islas.
m ó Num_{sol}	Número de soluciones a enviar, conocida como número de emigrantes en el modelo islas.
S	Envío de migrantes.
R	Reemplazo de migrantes.

Glosario de Términos

Plataforma Grid

Clúster	Conjunto de computadoras unidas a través de una red local de alta velocidad habilitadas para trabajar cómputo paralelo.
Grid	Conjunto de clúster de computadoras ubicadas en diferentes lugares geográficos y unidos mediante una red global para permitir la ejecución de aplicaciones paralelas.
Homogéneo	Término que se utiliza para indicar que todos los componentes de un sistema tienen características idénticas, es decir, son de un solo tipo.
Heterogéneo	Término que se utiliza para describir que los componentes de un sistema tienen diferentes características de capacidades de memoria, velocidades, entre otras.
Monoprocesador	Computadora con un solo procesador
Multiprocesador	Computadora paralela con múltiples procesadores que comparten un espacio de direcciones.
Memoria Compartida	Término que se aplica para indicar la presencia de una región de memoria que es compartida entre los componentes de un sistema. En cómputo paralelo significa que la memoria es compartida entre los procesos.
Memoria Distribuida	Término utilizado para hacer referencia a cientos o miles de computadoras independientes que al unirse forman una computadora paralela.
Ancho De Banda	En cómputo paralelo, se refiere al número de bytes por segundo que pueden transferirse en la red. Un programa paralelo que envía paquetes grandes puede ser limitado por el ancho de banda de la red.
Latencia	En cómputo paralelo, se refiere al tiempo que tarda un paquete de información en llegar desde el origen hasta el destino.
Eficiencia	Rapidez con que se encuentran las soluciones.
Eficacia	Calidad de las soluciones..

Cómputo Paralelo

Nodo	Elementos de cómputo que constituyen una máquina paralela de memoria distribuida. Cada nodo tiene su propia memoria y al menos un procesador, de tal forma que puede ser monoprocesador o multiprocesador.
Proceso	Es la entidad dinámica que representa la ejecución de un programa sobre un procesador. Colección de recursos que habilitan la ejecución de las instrucciones de un programa computacional. Esos recursos pueden incluir memoria virtual, descriptores de entrada/salida, identificadores de usuario y de grupo. El proceso tiene inicio, desarrollo y fin y un estado que evoluciona con el tiempo.
Sincronización	Procedimiento que se utiliza con el fin de garantizar que el acceso, de diferentes procesos, a los recursos compartidos sea adecuado para mantener los datos consistentes.

Teoría de Grafos y Redes

Grafo	Representación gráfica, compuesta por vértices y arcos, que permite modelar problemas de redes.
Arco o Arista	Elemento de un grafo, que se conecta de un nodo origen a un nodo destino. En el problema de Diseño de Redes de Distribución de Agua representa a las tuberías de la red.
Vértice o nodo	Elemento de un grafo que en el problema de Diseño de Redes de Distribución de Agua representa, entre otros, a los usuarios de la red.
Circuito	Camino que se forma en un grafo iniciando a partir de un nodo origen y finalizando al llegar nuevamente a ese nodo origen, formando así una sucesión de vértices.

Algoritmo Evolutivo

Configuración	Representación de la red que contiene diámetros de tuberías asignados a las tuberías de la red.
Individuo	Es sinónimo del término solución, pero se menciona al hablar del algoritmo evolutivo.
Solución	Configuración que cumple restricciones del modelo, en Algoritmos Evolutivos comúnmente se le conoce como Individuo Factible.

Individuo Factible	Solución que cumple las restricciones del modelo matemático definido en este trabajo.
Solución Óptima	Solución Factible y de menor costo
Espacio de Soluciones	Conjunto de soluciones factibles.
Espacio de Búsqueda	Conjunto de todas las posibles configuraciones.
Algoritmo Evolutivo	Método computacional que se basa en la teoría de la Selección Natural.
Gen	Cada elemento del cromosoma de un individuo.
Genotipo	Representación del individuo mediante un conjunto de genes.
Fenotipo	Representación de los valores de los genes, valores reales para el problema abordado.
Población	Conjunto de individuos.
Factible	Solución para problema.
Infactible	Solución parcial para el problema, que no cumple con todas las restricciones del modelo definido.
Selección	Proceso para definir cuáles individuos pueden convertirse en padres.
Cruce	Combinación de cromosomas para generar nuevos individuos.
Probabilidad de Cruce	Valor definido para la combinación de individuos.
Mutación	Cambio de valor de uno o más genes de un individuo.
Probabilidad de Mutación	Valor de referencia definido para que un individuo pueda sufrir una mutación.
Operador de Mutación	Valor de referencia para que un gen de un individuo pueda mutarse.
Descendiente	Individuo resultante de la combinación de cromosomas.
Aptitud	Valor que define la calidad de un individuo, en base a la función objetivo del problema.
Función Objetivo	Es una medida del logro de un objetivo. Ejemplos comunes de objetivos, en programación lineal, son maximizar ganancias o minimizar costos.
Generación	Iteración del algoritmo evolutivo.

Modelo de Paralelización

Modelo Maestro/Esclavo	Modelo para cómputo paralelo que se basa en un grupo compuesto por un proceso maestro y uno o más esclavos.
Maestro	Proceso que se encarga de asignar y coordinar a los esclavos.
Esclavo	Proceso que se encarga de realizar el trabajo asignado por el maestro.
Modelo Islas	Modelo definido en grupos independientes de procesos que pueden o no interactuar.
Emigrantes	Individuos que salen de una isla.
Inmigrantes	Individuos que llegan a una isla.
Migrantes	Individuos que llegan a una isla o salen de ella.
Modelo Maestro/Alumnos	Modelo híbrido compuesto por el modelo maestro-esclavos y modelo de islas.
Profesor	Es el equivalente al maestro en modelo maestro esclavos, asigna tareas y coordina a los alumnos.
Alumnos	Son el equivalente a los esclavos en el modelo maestro-alumnos. Realizan tareas asignadas y envían tareas respetando reglas de evaluación.
Reglas de Evaluación	Son las decisiones tomadas por el maestro para definir el tiempo en los que los alumnos deben entregar tareas y la forma en la que las tareas deben ser entregadas.
Tiempo de Entrega	Momento en que el alumno envía tareas al maestro, tomando como referencia las iteraciones del algoritmo.

Capítulo 1. Introducción

En este capítulo, se presenta un panorama general sobre Optimización Combinatoria y técnicas existentes para solucionar problemas de gran complejidad. Se presenta una introducción general al problema de Diseño de Redes de Distribución de Agua, que se aborda en esta investigación. Así mismo, se describen de manera general, las técnicas heurísticas que se han desarrollado para dar solución al problema y las plataformas de cómputo paralelo utilizadas. Adicionalmente, se describe la motivación de la cual surge este trabajo de investigación, los objetivos planteados inicialmente y los objetivos que se alcanzan al finalizarlo.

1.1. Introducción a la Optimización Combinatoria

La optimización combinatoria es una de las áreas de estudio de gran interés en la comunidad científica, por la diversidad de aplicaciones prácticas en la vida real.

Uno de los principales objetivos de la optimización combinatoria es encontrar “la mejor solución” posible para un problema de optimización. La mejor solución se encuentra dentro de un conjunto finito de configuraciones, mejor conocido en la literatura como espacio de búsqueda o espacio de soluciones. Aquí se le llama únicamente espacio de búsqueda porque contiene configuraciones que pueden o no ser soluciones para el problema a resolver. El espacio de soluciones, que se menciona a lo largo de este documento, se refiere al conjunto de configuraciones que solucionan el problema. Así, puede decirse que el espacio de soluciones es un subconjunto del espacio de búsqueda.

Desde el punto de vista matemático, un problema de optimización se caracteriza por tener una función objetivo a optimizar, un espacio de búsqueda y un conjunto de posibles soluciones. Generalmente, la solución que se desea encontrar, dentro de un conjunto finito de posibilidades, puede ser un número natural, una permutación o una estructura de grafo [Papadimitriou, 1998].

La calidad de una solución se determina principalmente, con base en la capacidad de minimizar o maximizar a la función objetivo. Así, para un problema de minimizar la mejor solución ***s*** es aquella solución que representa el menor costo, mientras que para un problema de maximizar la mejor solución ***s*** es aquella que representa el mayor costo. En un problema de minimizar puede observarse la existencia de varios valores conocidos como mínimos locales, que pueden ser soluciones válidas para el problema, pero sólo existe un valor conocido como mínimo global, el cual se refiere a la mejor solución. Es decir, el mínimo global es la solución que presente el menor costo. De manera similar, en un problema de maximizar puede observarse la existencia de varios valores conocidos como máximos locales y la existencia de un valor conocido como máximo global, que es la solución de mayor costo.

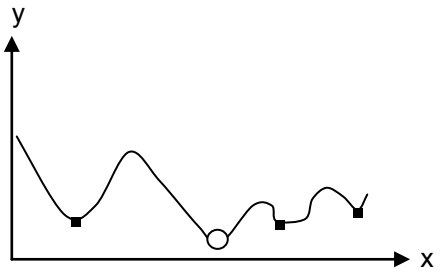


Figura 1.1 Problema de Minimizar

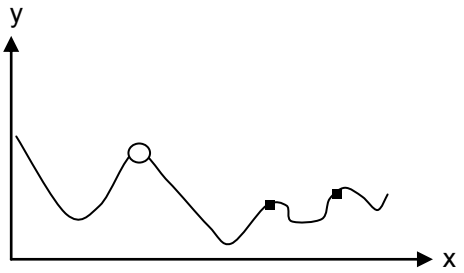


Figura 1.2 Problema de Maximizar

Las Figuras 1.1 y 1.2 muestran la representación gráfica de un problema de minimizar y maximizar, respectivamente. En el eje de las x se encuentra representado el espacio de posibles soluciones, para un problema de optimización combinatoria. En el eje de las y, se encuentra representada la función objetivo, ya sea minimizar o maximizar. En la Figura 1.1 puede verse que, para el problema de minimizar, existen varios mínimos locales representados con puntos y un mínimo global representado con un círculo. De manera similar, en la Figura 1.2 puede verse, que para un problema de maximizar, existen varios máximos locales representados con puntos y un máximo global representado con un círculo. Al valor mínimo global y máximo global normalmente se les conoce también como óptimo global. Estos valores, que representan a la "mejor solución", son precisamente los que se buscan cuando se resuelve un problema de optimización.

Los problemas de optimización han sido estudiados con base en su dificultad de resolución. Existe un área de estudio, llamada Teoría de la Complejidad, que clasifica a

los problemas en diferentes categorías de acuerdo con el tiempo que tarda una técnica computacional en encontrar una solución válida para un problema. Así, dependiendo de la complejidad computacional que conlleva la resolución de un problema [Cook,1971], éste puede ubicarse en diferentes categorías conocidas como clase P, clase NP y clase NP-Completos, Figura 1.3.

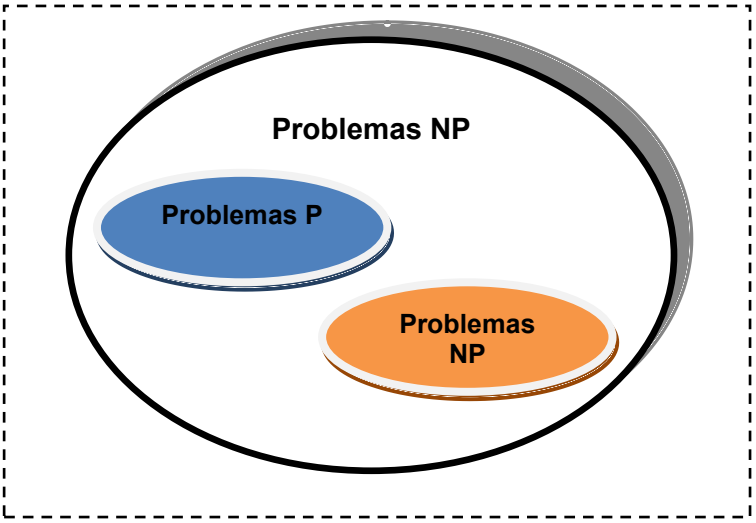


Figura 1.3 Clasificación de Problemas

La clase P (Deterministic Polinomial) está compuesta por problemas de decisión, tratables, que se resuelven con técnicas computacionales conocidas como algoritmos deterministas[1], en tiempo polinomial [Stinson, 1987]. Es decir, para entradas de tamaño *n* el tiempo de resolución es O(*nk*) para alguna constante *k*, en el peor de los escenarios. Esto significa que los problemas de la clase P pueden ser resueltos por algoritmos deterministas en un número de pasos acotado por un polinomio fijo en función del tamaño de la entrada. Aun cuando las instancias[2] se incrementen en tamaño, el problema de clase P puede ser resuelto de manera eficiente mediante el uso de métodos exactos [Cormen, 2001].

La clase NP (Non Deterministic Polinomial) corresponde a problemas que hasta la fecha [Cruz, 2010a] no han podido ser resueltos de manera exacta por medio de algoritmos deterministas. Se dice que estos problemas son fáciles de verificar pero no son fáciles de resolver [Papadimitriou, 1998].

[1] Procesos en los que dada una entrada, se produce siempre la misma salida

[2] Archivo de entrada para el algoritmo que contiene la definición explicita del problema.

La clase NP-Completos (Complete-Non Deterministic Polinomial) contiene a problemas que son muy difíciles de resolver en tiempo polinomial, principalmente porque el espacio de posibles soluciones crece de manera exponencial junto con el tamaño de la entrada. Algunos ejemplos que se encuentran dentro de esta categoría son el problema del agente viajero (TSP, Travelling Salesman Problem) [Gutin, 2004], problemas de planificación de tareas (Scheduling Problems) [Cruz, 2014b], el problema de Redes de Distribución de Agua (Water Distribution Network Design) [Gupta, 1993], entre otros.

Las estrategias para resolver problemas de optimización pueden clasificarse en diferentes categorías. De manera general, puede decirse que existen técnicas de solución conocidas como métodos exactos y técnicas de solución conocidas como métodos de aproximación. Los problemas que se encuentran en la clase P generalmente se resuelven de forma óptima y en un tiempo de cómputo razonable mediante el uso de métodos exactos (algoritmos voraces, algoritmos de ramificación y poda, algoritmos de divide y vencerás, entre otros). Los métodos exactos aseguran la obtención del óptimo global para resolver problemas de la clase P. Sin embargo, para solucionar problemas que pertenecen a la clase NP-Completos estos métodos pueden consumir tiempo computacional excesivo. Por ejemplo, para una instancia de prueba del problema de Diseño de Redes de Distribución de Agua, con 20 tuberías y un conjunto discreto de 10 diámetros de tuberías a elegir, el espacio de búsqueda para el problema equivale a 10^{20} diseños diferentes. Aun cuando se realizaran 1 000 000 de operaciones por segundo, se necesitarían 3 000 000 de años para generar todos los diseños mediante un método de enumeración. De acuerdo con lo anterior, la aplicación de métodos exactos en problemas de gran tamaño suele ser inviable y es precisamente en estos casos cuando se recurre al uso de heurísticas, cuyo término generalmente se usa en contraposición al término exacto [Melián, 2003].

La palabra heurística proviene de la misma rama que la palabra griega *Eureka*, que significa “lo he encontrado*”*. De forma genérica, se puede definir como el arte y la ciencia de descubrir e inventar. Las heurísticas, también conocidas como métodos heurísticos, son técnicas basadas en la experiencia, la repetición y la generación de reglas empíricas que permiten encontrar soluciones de alta calidad, con un costo computacional razonable. Son “Procedimientos simples, a menudo basados en el sentido común, que se supone

que obtendrán una buena solución, no necesariamente óptima, de un modo sencillo y rápido para problemas difíciles" [Zanakis, 1981]. Las heurísticas pueden utilizarse en problemas de optimización que presentan espacios de búsqueda muy grandes y/o complejos. Estas técnicas, buscan aproximarse de manera intuitiva, pero ordenada, a la mejor solución y se ha comprobado empíricamente que estos métodos funcionan muy bien para cierto tipo de problema. La ventaja principal de los métodos heurísticos es que éstos encuentran soluciones válidas para problemas de optimización en un tiempo de cómputo finito. Sin embargo, estos métodos no garantizan que de las soluciones encontradas alguna de ellas sea la solución óptima, e incluso en algunas ocasiones no se establece que tan factible es una solución.

Los métodos heurísticos son una alternativa cuando no es posible utilizar métodos exactos. No obstante, cuando es indispensable asegurar que la solución encontrada sea precisamente la solución óptima las heurísticas no son recomendables. Los métodos heurísticos tienen su principal limitación en su incapacidad para escapar de óptimos locales, por lo que en ocasiones presentan soluciones cuasi óptimas como óptimas. La incapacidad de escapar de óptimos locales ocurre generalmente, cuando los algoritmos heurísticos no utilizan algún mecanismo que les permita proseguir la búsqueda del óptimo en el caso de quedar atrapados en óptimos locales.

En los años sesenta, surgió un nuevo tipo de algoritmo de aproximación para solucionar las limitaciones que presentaban los métodos heurísticos. En estos algoritmos de aproximación, la idea central se enfocaba en realizar una exploración del espacio de búsqueda de una manera más eficiente y eficaz. A estos algoritmos se les denominó Metaheurísticas. Las Metaheurísticas son algoritmos de búsqueda más inteligentes que los heurísticas [Glover, 1986]. Son procedimientos de alto nivel que guían a métodos heurísticos conocidos, evitando que éstos queden atrapados en óptimos locales. Son una clase de métodos de aproximación que están diseñados para resolver problemas difíciles de optimización combinatoria (problemas NP) en los que los heurísticos clásicos no son efectivos, [Kelly, 1996]. Las Metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos". Son procesos iterativos que guían y modifican las operaciones de una heurística subordinada para producir

eficientemente soluciones de alta calidad [Voss, 1999]. Las metaheurísticas pueden manipular una única solución completa (o incompleta) o una colección de soluciones, en cada iteración. La heurística subordinada puede ser un procedimiento de alto o bajo nivel, una búsqueda local, o un método constructivo. La tabla 1.1 muestra una comparación general de las técnicas de optimización mencionadas.

Tabla 1.1 Algoritmos de Optimización

Método	Características	Ventajas	Desventajas
Exacto	Adecuados para problemas P	Obtienen el óptimo global	Consumen gran cantidad de recursos de tiempo y espacio para instancias consideradas grandes
Aproximación	Adecuados para problemas NP	Encuentran soluciones de calidad. Obtienen soluciones en tiempos razonables.	No existe forma de verificar que la solución encontrada sea la óptima

En general, las técnicas heurísticas se ajustan de manera fiel a un problema de optimización concreto y son muy útiles para abordar problemas donde los espacios de búsqueda son grandes y/o complejos, tal como lo es el espacio de búsqueda del problema de Diseño de Redes de Distribución de Agua [Baños, 2006]. Las heurísticas son consideradas como una de las mejores opciones para abordar problemas NP-Completos porque prometen encontrar buenas soluciones en tiempos de cómputo razonables. Ésta es la razón principal que justifica el uso de técnicas heurísticas en el desarrollo de esta tesis, enfocándose así en el estudio e implementación de una heurística conocida bajo el nombre de Algoritmo Evolutivo.

Los Algoritmos Evolutivos (AE) se encuentran dentro del paradigma de la computación evolutiva [Holland, 1975]; son algoritmos de búsqueda estocástica que se basan en la abstracción de los procesos definidos en la teoría de la evolución de Darwin

[Darwin, 1872], cuyo principal argumento es que los individuos más aptos son los que deben sobrevivir y generar una descendencia más facultada basándose en la teoría de la evolución natural de los seres vivos. La idea principal de esta teoría es que los individuos que tengan una mejor adaptación al medio tendrán mayores probabilidades de vivir y de generar descendientes que hereden sus características. Por el contrario, los individuos con peor adaptación al medio tendrán menores probabilidades de sobrevivir y generar descendientes, por lo que probablemente se extingan sin dejar descendencia.

Los algoritmos evolutivos, son algoritmos de naturaleza robusta que realizan una búsqueda global en un espacio de búsqueda constituido por un conjunto de posibles configuraciones para solucionar un problema. Estos algoritmos se han aplicado con éxito, a una gran variedad de problemas, en diferentes áreas de investigación. A pesar de que no garantizan encontrar la solución óptima para un problema, existen diversos estudios experimentales [Baños, 2006], [Cruz, 2010], que muestran la eficiencia, de dichos algoritmos, para encontrar soluciones para problemas NP-Completos. Sin embargo, un factor que debe considerarse al utilizar algoritmos evolutivos, es el alto costo computacional que implica su implementación al abordar problemas de optimización reales, de gran tamaño y complejidad [Reca, 2006]. El alto costo de cómputo, se genera principalmente con el ajuste de los diferentes parámetros que tienen dichos algoritmos y con las búsquedas exhaustivas que realizan para encontrar una solución adecuada al problema. Encontrar la solución óptima para un problema de optimización real, de gran tamaño, puede resultar una tarea costosa en cuanto a tiempo y espacio computacional. Ante este problema, una alternativa es recurrir a la implementación de métodos heurísticos en ambientes que habilitan el uso de cómputo paralelo.

En la actualidad, es posible disponer de diferentes tipos de plataformas para la computación de alto rendimiento. Por ejemplo, los clúster de computadoras pueden ayudar a reducir el tiempo de cómputo necesario para solucionar un problema, ya que permiten disponer de una gran cantidad de recursos computacionales. Los clúster de computadoras son un conjunto de computadoras interconectadas, con tecnologías de red de alta velocidad, que trabajan de forma conjunta para alcanzar un objetivo. Los clúster de computadoras, comparten recursos como procesadores, memoria, servicios, entre otros.

Estas plataformas permiten solucionar un problema en menor tiempo. Los clúster de computadoras pueden unirse formando plataformas Grid.

La Grid es una infraestructura de software y hardware heterogéneos que mediante el uso de la red permite compartir recursos de procesamiento y de almacenamiento para solucionar problemas en ambientes distribuidos a través de la ejecución de aplicaciones paralelas. Se puede acceder a recursos computacionales de alto nivel de forma consistente y a bajo costo [Tsaregorodtsev, 2009].

La implementación de aplicaciones paralelas se lleva a cabo, principalmente, para resolver problemas complejos [Blazewicz, 2000] en los que los recursos de una sola máquina no son suficientes. La finalidad del cómputo paralelo es disminuir el tiempo de procesamiento mediante la distribución de tareas entre los procesadores disponibles. La paralelización de algoritmos se basa en el concepto de cooperación, por lo que diversos procesos trabajan en conjunto para reducir el tiempo para solucionar un problema y para mejorar también la calidad de las soluciones de un problema. Actualmente, la paralelización de algoritmos es una técnica que se utiliza comúnmente cuando se abordan problemas de optimización de tipo NP-Completos, como el problema de Diseño de Redes de Distribución de Agua.

El problema de Redes de Distribución de Agua (RDA), es un problema de gran interés para los investigadores por su amplia aplicación práctica. Durante más de tres décadas ha sido estudiado ampliamente. Se han propuesto diferentes formulaciones matemáticas y un gran número de métodos de solución [Cruz, 2009a]. No obstante, en la práctica sólo se han resuelto instancias pequeñas debido a la enorme complejidad del problema [Baños, 2006], mismo que de acuerdo con la teoría de la complejidad se clasifica dentro de los problemas NP-Completos [Cruz, 2009b].

De forma general el problema, de Diseño de Redes de Distribución de Agua, consiste en elegir los componentes básicos que forman parte de la red de tal forma que la red resultante sea una red de costo mínimo. Entre los componentes más comunes para el diseño de la red se encuentran: tuberías, bombas, válvulas y fuentes de abastecimiento. Las tuberías se encuentran disponibles, por diferentes comercios, en diferentes diámetros

y materiales. La función que desempeñan en una red de distribución de agua es la de llevar el vital líquido desde la fuente hasta los usuarios de la red. Las válvulas reguladoras, son elementos que ayudan a modular la presión obtenida en una red de distribución de agua. Las bombas de potencia son elementos indispensables, en caso de que la técnica de distribución del agua sea bombeo. Finalmente, las fuentes de abastecimiento son elementos imprescindibles en la red de distribución de agua y pueden ser ríos, arroyos, manantiales, pozos, entre otros. De acuerdo con [Bhave, 1991], las redes de distribución de agua pueden clasificarse, de acuerdo con la topología que presenten, en redes en serie, redes ramificadas y redes malladas.

Una red en serie es aquella que no contiene mallas ni ramificaciones; es una conexión entre dos o más nodos de forma lineal. Generalmente, tienen un nodo fuente, un nodo final y uno o más nodos intermedios. Ésta, es la topología más simple que existe para las redes de distribución de agua.

Una red ramificada es un conjunto de redes en serie y no contiene mallas. Estas redes presentan una estructura similar a la estructura de un árbol, presentan un nodo fuente, más de un nodo final y uno o más nodos intermedios. Generalmente, son redes que se utilizan para la distribución de agua en pequeñas comunidades rurales, zonas industriales o en zonas de riego. En la práctica, estas redes presentan como inconveniente la suspensión del servicio en diferentes puntos de la red cuando ocurren roturas o fugas en alguna tubería. Esto se debe a que en las redes ramificadas, sólo existe un camino para llegar de un punto a otro en la red. Por esta razón, algunos usuarios se ven afectados con la suspensión del servicio cuando ocurren fallas en la red.

Las redes malladas son redes que contienen circuitos o mallas, lo cual significa que para un usuario el agua puede llegar a través de diferentes caminos. En estas redes, la interrupción del servicio ocasionado por rupturas en las tuberías ocurre con menor frecuencia, ya que el agua puede llegar a su destino utilizando diferentes trayectorias. Por esta razón, una rotura en una tubería, generalmente, no afecta gravemente a otros puntos de la red, así que en estas redes es menos frecuente que los usuarios tengan fallas en el servicio. A pesar de que es más costoso implementar las redes malladas que las redes ramificadas, se justifica la implementación porque las redes malladas ofrecen mayor

confiabilidad. El empleo de estas redes es habitual en redes de distribución urbanas, redes de riego, en donde se exige al sistema una gran seguridad en el suministro de agua. La investigación realizada en este trabajo de tesis se enfoca precisamente en el problema de diseño de redes de distribución malladas, utilizando para la solución de dicho problema un algoritmo evolutivo en ambiente Grid, trabajando en conjunto con un simulador hidráulico llamado Epanet.

1.2. Objetivo General

El objetivo principal de este trabajo de tesis es desarrollar un algoritmo evolutivo en ambiente Grid para encontrar soluciones de alta calidad (soluciones que cumplen todas las restricciones de los dos modelos matemáticos definidos en el apartado 3.3) en un tiempo de cómputo aceptable para el problema de tipo NP-Completo, conocido como el problema de Diseño de Redes de Distribución de Agua.

1.3. Objetivos Específicos

1. Formular un modelo matemático que represente el diseño de una red de distribución de agua en el que el objetivo es brindar a los usuarios de la red un buen servicio (tanto en la operación de la red como en el servicio de los usuarios), con presiones mínimas requeridas para satisfacer sus demandas hídricas.
2. Implementar un algoritmo evolutivo en ambiente Grid para solucionar el modelo matemático y obtener soluciones de calidad para el problema de Diseño de Redes de Distribución de Agua.
3. Realizar estudios experimentales para probar que el método de solución propuesto es eficaz y eficiente para el problema de Diseño de Redes de Distribución de Agua.
4. Comparar la eficiencia y la eficacia del algoritmo evolutivo paralelo con versiones propias de algoritmos secuenciales.
5. Comparar la eficiencia y la eficacia del algoritmo evolutivo paralelo con los resultados obtenidos por algoritmos de diferentes investigadores, reportados en la literatura.

1.4. Planteamiento del Problema

El problema de Diseño de Redes de Distribución de Agua ha sido formulado matemáticamente, con diferentes enfoques, por diversos investigadores. De forma general, el problema consiste en minimizar el costo del diseño de una red de distribución de agua garantizando, de forma implícita, que con el diseño de dicha red los usuarios tengan un buen servicio y puedan satisfacer sus demandas hídricas. Sin embargo, en ocasiones las redes de distribución de agua ya diseñadas pueden presentar problemas de diseño que afectan el funcionamiento de la red. Algunas redes, principalmente ramificadas, ofrecen servicios deficientes en donde los usuarios no tienen las presiones mínimas requeridas para obtener los recursos hídricos de la red. Además, con el paso del tiempo las tuberías existentes pueden tener diámetros que no satisfacen los requerimientos hídricos de los usuarios de la red, ya sea porque se definen inicialmente con diámetro incorrecto o porque éstos se reducen considerablemente con las sedimentaciones o por problemas de raíces de árboles que aprisionan la tubería, estrangulándola y disminuyendo su diámetro, ocasionando que el servicio de distribución de agua en la red sea ineficiente. Ante esto, la red necesita rediseñarse total o parcialmente para ofrecer un servicio adecuado a los usuarios. El proceso de optimización consiste en obtener la configuración de una red de costo mínimo que sirva como diseño definitivo para aplicarse de forma práctica al implementar una red de distribución de agua. La configuración se refiere a un listado de diámetros de tuberías óptimos para la red.

Para poder alcanzar los objetivos planteados previamente se han definido las siguientes etapas con los pasos del método científico.

1. Observación. Se realizó un estudio exhaustivo del problema de optimizaciones del Diseño de Redes de Distribución de Agua, en instancias teóricas y una práctica definidas en 3.2, para analizar el problema y las técnicas que se han aplicado para la resolución del mismo.
2. Definición de un modelo matemático que representa al problema de Diseño de Redes de Distribución de Agua. En este trabajo, el problema ha sido representado a través de dos modelos, un modelo de programación lineal y un modelo de satisfacción de restricciones.

3. Estudio bibliográfico para conocer los métodos de solución que han sido propuestos, por otros investigadores, para el problema en cuestión y así mismo conocer los resultados que se han obtenido con dichos métodos.
4. Elección de una heurística, basándose en el estudio definido en la etapa 3, para abordar de forma competitiva y exitosa el problema de Diseño de Redes de Distribución de Agua.
5. Definición de hipótesis de investigación.
    - ¿Existe menor cantidad de soluciones factibles que infactibles en el espacio de búsqueda del problema RDA?.
    - ¿Un óptimo local es alcanzado cuando existe poca diversidad en una población?.
    - ¿La calidad de la población inicial ayuda al algoritmo a converger más rápidamente?.
    - ¿Un individuo factible puede obtenerse a través de la combinación de dos individuos factibles, de dos individuos infactibles o bien de la combinación de individuos factibles con infactibles?.
    - ¿Las poblaciones compuestas únicamente por los mejores descendientes pueden ayudar al algoritmo a encontrar soluciones de alta calidad para el problema Redes de Distribución de Agua?.
6. Estudio y experimentación del espacio de búsqueda del problema RDA para la instancia de prueba Two-Looped Network [Alperovits, 1977] definida en la literatura.
7. Estudio y experimentación de los operadores de los algoritmos evolutivos clásicos: selección, cruce y mutación.
8. Estudio y experimentación de la población inicial del algoritmo evolutivo, tomando para ello muestras representativas de diferente tamaño, para la instancia de prueba Two-Looped Network.
9. Diseño de un algoritmo evolutivo novedoso funcionando en ambiente grid, bajo la implementación de modelo de paralelización propio véase 4.3.3, que permite abordar el problema de Diseño de Redes de Distribución de Agua y resolver instancias definidas y utilizadas en la literatura, con la finalidad de obtener soluciones de calidad en tiempos de cómputo limitados.

10. Ajuste de los parámetros del algoritmo (Tamaño de Población, Número de Generaciones, Probabilidad de cruce, Probabilidad de Mutación), para saber cuáles son los mejores valores que permiten obtener las mejores soluciones para el problema de Diseño de Redes de Distribución de Agua.
11. Análisis de los resultados obtenidos con el algoritmo evolutivo desarrollado en este trabajo (con las instancias Two-Looped Network, Hanoi y Balerma) para realizar una comparación con los resultados obtenidos por los métodos de solución propuestos por otros investigadores.
12. Codificación e implementación de un novedoso algoritmo evolutivo en ambiente Grid para dar solución al problema de Diseño de Redes de Distribución de Agua. El desarrollo de este algoritmo se basa en los resultados obtenidos por toda la experimentación previamente realizada, definida en las etapas anteriores. El algoritmo desarrollado, es implementado basándose en los resultados experimentales de las hipótesis propuestas. La integración de los resultados en la implementación del algoritmo permiten generar un algoritmo evolutivo robusto que obtiene resultados de calidad.
13. Comparación y análisis de los resultados para mostrar la calidad de las soluciones encontradas y el tiempo requerido para la ejecución de diferentes instancias de prueba consideradas pequeñas, medianas y grandes.
14. Obtención de conclusiones, escritura de artículos y del documento de tesis.

1.5. Alcances

Para el problema de Diseño de Redes de Distribución de Agua, se tienen contemplados los siguientes alcances.

- Definición de un modelo matemático que represente al problema de Diseño de Redes de Distribución de Agua.
- Solución al modelo matemático mediante un Algoritmo Evolutivo.
- Diseño e implementación de un Algoritmo Evolutivo Secuencial para el problema de Diseño de Redes de Distribución de Agua.
- Diseño e implementación de un Algoritmo Evolutivo funcionando en ambiente Grid para el problema de Diseño de Redes de Distribución de Agua.

- Solución al problema de Diseño de Redes de Distribución de Agua utilizando instancias teóricas para comparar resultados con otros investigadores.
- Solución al problema de Diseño de Redes de Distribución de Agua para un caso de estudio real.

1.6. Justificación

El suministro óptimo de los recursos hídricos para los usuarios de una red de distribución de agua es un tema que está siendo cada vez más estudiado debido a la escasez del agua y a la importancia vital que ésta tiene para los usuarios.

El diseño de una red de distribución de agua es un aspecto muy importante que ayuda a garantizar el funcionamiento adecuado de la red de distribución de agua, para permitir que los usuarios de la red tengan un servicio que satisfaga sus necesidades hídricas. Por esta razón, es necesario encontrar modelos empíricos o teóricos que, basados en las leyes de la hidráulica, reflejen el comportamiento de la red y permitan estudiar el diseño y la operación de la misma antes de emprender un proyecto. Estos modelos son de gran utilidad para poder minimizar el costo del diseño de una red de distribución de agua y al mismo tiempo tener una operación adecuada de la red para satisfacer las necesidades hídricas de los usuarios, mediante simulaciones previas de la red. Los programas computacionales permiten conocer de forma temprana el diseño de la red, el costo de la misma y el funcionamiento previo a su implementación.

La principal motivación de este trabajo fue poder contribuir con la elaboración de redes de distribución de agua con el menor costo posible y al mismo tiempo el mejor servicio al usuario, ya sea en redes nuevas o en redes existentes.

1.7. Contribuciones

Las principales contribuciones de este trabajo doctoral son:

- Modelado del problema de Diseño de Redes de Distribución de Agua.
- Diseño y desarrollo de un novedoso algoritmo evolutivo que trabaja en ambiente Grid para abordar el problema de Diseño de Redes de Distribución

de Agua en una instancia real de gran tamaño llamada Balerma. Esta contribución es importante, puesto que hasta la fecha no se ha desarrollado un algoritmo similar para el problema abordado.

- Desarrollo de un Analizador de resultados que de forma independiente al algoritmo permite la interpretación confiable de los resultados.
- Estudio de los componentes del Algoritmo Evolutivo, mediante un Algoritmo Paralelo que permite abarcar un estudio mayor que con el Algoritmo Secuencial en un tiempo de cómputo limitado.
- Propuesta e implementación de un modelo híbrido llamado maestro-alumnos que combina las bondades del modelo maestro-esclavos y el modelo de islas.
- Resultados de calidad. De acuerdo con la investigación realizada, puede decirse que el algoritmo desarrollado en este trabajo doctoral ha conseguido mejorar los resultados publicados en otros trabajos en cuanto a eficiencia y eficacia, véase capítulo seis.

1.8. Organización de la Tesis

La estructura de este trabajo de tesis consiste en siete capítulos principales donde se describe el contenido de la investigación. Adicionalmente, contiene un capítulo de referencias de distintos trabajos que se mencionan a lo largo del documento de tesis. Contiene también cuatro apéndices en el que se describen: 1) Instancias de prueba utilizadas 2) Código fuente del Algoritmo Evolutivo 3) Detalles de la Infraestructura utilizada y 4) Publicaciones resultantes de este trabajo de investigación, a las cuales se hace alusión y fundamentan el contenido de este trabajo doctoral. A continuación se describe brevemente a cada uno de los capítulos:

Capítulo 1. Introducción. En este capítulo se presenta una introducción general sobre el problema que se aborda en esta investigación. Se presentan conceptos introductorios sobre el problema de Diseño de Redes de Distribución de Agua. También se describen, de manera general, los métodos existentes para abordar el problema y las herramientas computacionales que sirven de apoyo. Finalmente, se describe la motivación para el desarrollo de este trabajo y los objetivos que se alcanzan al finalizarlo.

Capítulo 2. Antecedentes. En este capítulo se presenta una revisión detallada de los trabajos, más relevantes, relacionados con el problema de Diseño de Redes de Distribución de Agua, que han sido publicados por diferentes investigadores a lo largo de tres décadas. Los trabajos consultados sirven como punto de partida para esta investigación doctoral. Al mismo tiempo, estos trabajos permiten comparar el trabajo desarrollado en esta tesis con trabajos desarrollados por otros investigadores. El estudio de antecedentes históricos presentado en este capítulo se enfoca en aquellas publicaciones cuyo método de solución se basa en la implementación las técnicas conocidas como algoritmos evolutivos.

Capítulo 3. Fundamentos del problema RDA. Después de finalizar el estudio de los trabajos relacionados con el problema de Diseño de Redes de Distribución de Agua, en este capítulo se muestra una descripción detallada de dicho problema. Se presentan los fundamentos teóricos del problema y la formulación matemática definida para el problema de Diseño de Redes de Distribución de Agua estudiado en este trabajo doctoral. Finalmente, se describen las instancias de prueba que se resuelven.

Capítulo 4. Método de solución para el problema de Diseño de Redes de Distribución de Agua. En este capítulo se presenta el método de solución propuesto para el problema de Diseño de Redes de Distribución de Agua. El método de solución es un algoritmo evolutivo que se explica detalladamente, mediante escenarios basados en instancias teóricas (*benchmarks)* que han sido utilizadas en varios trabajos previos desarrollados por diferentes investigadores.
Se describen también, estudios experimentales acerca del espacio de búsqueda, de la población inicial, de los operadores del algoritmo evolutivo, de los valores paramétricos del mismo, entre otros. Estos estudios han sido realizados durante este trabajo doctoral, se han probado de forma independiente y se han implementado en la versión final del algoritmo, dando resultados favorables.

Capítulo 5. Diseño del Algoritmo Evolutivo para el problema RDA. En este capítulo se presenta la etapa del diseño del Algoritmo Evolutivo, desarrollado en este trabajo doctoral. Se presentan las versiones finales tanto del algoritmo evolutivo secuencial como del algoritmo evolutivo paralelo, que han permitido obtener soluciones de calidad para el

problema de Diseño de Redes de Distribución de Agua, abordado en esta tesis. Así mismo, se describen los ajustes paramétricos en las versiones finales del algoritmo, las características específicas y los detalles de implementación de los algoritmos.

Capítulo 6. Análisis de los resultados obtenidos. En este capítulo se describen los resultados obtenidos con el algoritmo evolutivo secuencial y con el algoritmo evolutivo paralelo funcionando en ambiente Grid. Se hace un análisis comparativo de los algoritmos secuencial y paralelo desarrollados para determinar la eficiencia y eficacia de ambos. También, se realiza una comparación con otros algoritmos desarrollados por diferentes investigadores utilizando para la comparación las mismas instancias de prueba.

Capítulo 7. Conclusiones. En este capítulo se presentan las conclusiones obtenidas al finalizar este trabajo de tesis doctoral. También, se presentan las contribuciones y aportaciones del trabajo desarrollado. Finalmente, se describen los trabajos futuros, que podrían dar continuidad a este trabajo de investigación.

Capítulo 2. Antecedentes

En este capítulo, se presenta una revisión bibliográfica de los trabajos, más relevantes, relacionados con el problema de Diseño de Redes de Distribución de Agua, que han sido publicados por diferentes investigadores. En especial, se hace un análisis de los artículos cuyo método de solución está basado en técnicas de algoritmos evolutivos.

2.1. Estado del Arte

A través del tiempo el problema de Diseño de Redes de Distribución de Agua ha sido estudiado ampliamente por diferentes investigadores; ha sido un problema de gran interés tanto para la sociedad, como para la comunidad científica debido a su amplia aplicación práctica. El problema de Diseño de Redes de Distribución de Agua no es un problema reciente. De hecho, éste ha sido estudiado desde hace aproximadamente tres décadas. Se considera que Alperovits y Shamir [Alperovits, 1977] son quienes proponen, inicialmente, el planteamiento del problema de un sistema de distribución de agua. A lo largo de este documento, se hace referencia a dicho planteamiento como enfoque clásico[3] del problema de Diseño de Redes de Distribución de Agua.

El planteamiento del enfoque clásico consiste en una red de tuberías a través de la cuales se lleva agua desde las fuentes hasta los usuarios. Además de las tuberías, la red puede incluir diferentes componentes. Algunos de ellos son bombas de potencia, válvulas, fuentes de abastecimiento, entre otros. En el enfoque clásico, las variables de decisión, para el problema de Diseño de Redes de Distribución de Agua, son los diámetros de las tuberías definidas comercialmente. Estas variables se definen como variables continuas. Adicionalmente, las capacidades de las bombas y la elevación de las fuentes de abastecimiento pueden ser consideradas como variables de decisión.

[3] Enfoque clásico es un término que se emplea en este trabajo para los trabajos iniciales, hace ya tres décadas.

Las restricciones que deben cumplirse en el problema RDA son las demandas hídricas de los usuarios y las presiones en los nodos de la red. Es importante mencionar que para cada par de nodos, que definen un segmento de la red, pueden existir dos o más tuberías con diferentes tamaños de diámetros. Las demandas hídricas de los usuarios pueden variar de acuerdo con el tipo de usuario que se tenga. Es decir, existen usuarios que pueden tener mayores cantidades de consumo que otros. Por ejemplo, si el usuario de la red es un área de cultivo, éste puede tener mayores requerimientos hídricos que un usuario común de la red, como personas. La operación adecuada de la red debe satisfacer las demandas de todo tipo de usuario, con base en la definición de presiones mínimas requeridas. Así, es necesario que se respeten límites mínimos y máximos de presiones para que exista un funcionamiento correcto de la red, evitando que con presiones inferiores al límite mínimo se acumulen sedimentos y con ello los diámetros de las tuberías se reduzcan. Así mismo, es importante asegurar que las presiones en la red no sean superiores al límite máximo permitido para garantizar que la red opere correctamente evitando rupturas en las tuberías ocasionadas por presiones muy elevadas. Respecto a los usuarios de la red, las presiones mínimas y máximas ayudan a tener un suministro de agua controlado acorde con las necesidades del usuario, garantizando un servicio adecuado y evitando desperdicios constantes por presiones muy elevadas.

Otro aspecto importante que debe mencionarse es que en el enfoque clásico los investigadores se enfocan en el estudio de redes de distribución de agua cuya topología es ramificada, y generalmente plantean el problema con un modelo de programación lineal (LP) y solo en algunos casos es formulado como modelo de programación no lineal (NLP). A pesar de que inicialmente las instancias de prueba que manejan son redes malladas, no tienen restricción en los diámetros de tubería que utilizan por lo que en ocasiones al utilizar diámetros de tuberías muy pequeños el resultado se interpreta como si fueran redes ramificadas. Finalmente, es importante decir que la formulación del problema es lineal y las técnicas de solución son generalmente métodos exactos.

La Tabla 2.1 muestra, con mayor detalle, la información del problema de Diseño de Redes de Distribución de Agua abordado por diferentes investigadores en el enfoque clásico del problema, para la instancia de prueba Alperovits.

Tabla 2.1 Problema RDA utilizando variables Continuas

Fecha	Autores	Método	Costo
1977	Alperovits et al.	Gradiente	497525
1979	Quindry et al.	Gradiente	441522
1986	Goulter et al.	Gradiente	435015
1987	Fujiwara et al.	Quasi-Newton	415271
1989	Kessler et al.	Gradiente	417500
1990	Loganathan et al.	Heurística	412931
1993	Gupta et al.	Fletcher-Powell	407625
1995	Loganathan et al.	Heurística	403561
1997	Varma et al.	Programación Cuadrática Sucesiva	441310

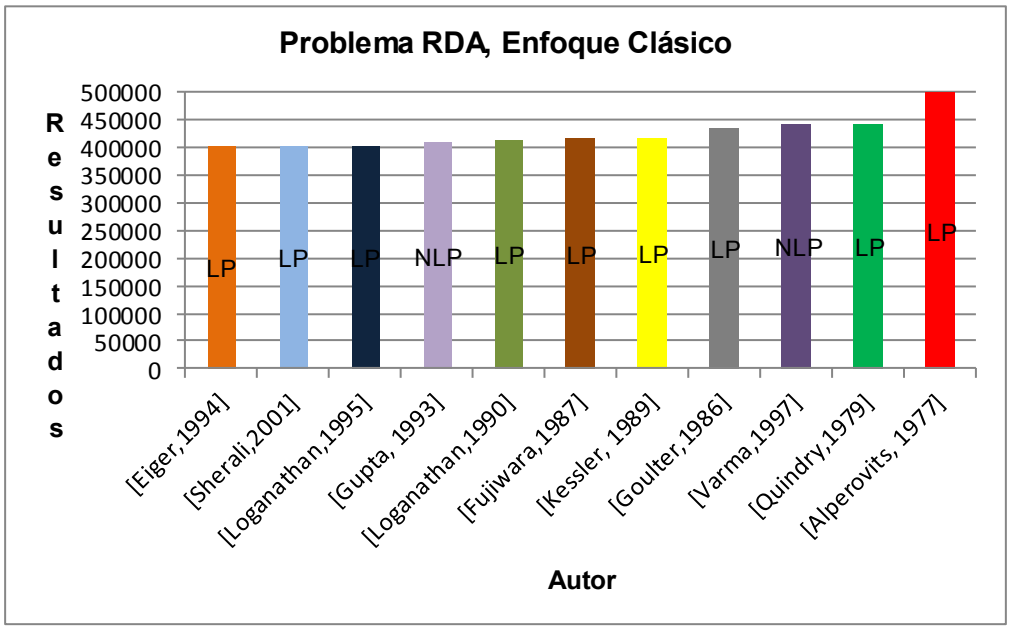


Figura 2.1 Comparación de los costos utilizando Variables Continuas

En su trabajo clásico, Alperovits propone al gradiente como método de solución. Después de Alperovits el problema de Diseño de Redes de Distribución de Agua ha sido estudiado por varios investigadores: Quindry [Quindry, 1979], Goulter et al [Goulter, 1986], Fujiwara et al. [Fujiwara, 1987], Kessler et al. [Kessler, 1989], Loganathan et al. [Loganathan, 1990], Gupta et al. [Gupta, 1993], Varma et al. [Varma, 1997], Sherali et al. [Sherali, 1998], entre otros. Estos investigadores proponen diferentes métodos de

solución que van desde modificaciones al método del gradiente propuesto por Alperovits et al., hasta nuevos métodos de solución como son Quasi Newton, Ramifica y Poda, entre otros. En el enfoque clásico, el problema de Diseño de Redes de Distribución de Agua se aborda bajo el mismo planteamiento. A pesar de que se proponen diferentes métodos de solución, los costos obtenidos por la mayoría de los investigadores son cercanos, Figura 2.1. Esto se debe principalmente a que la formulación del problema y las condiciones son similares. Así, puede decirse que la diferencia principal en estos trabajos es precisamente el método de solución que proponen para resolver el problema. Con el tiempo, el problema de Diseño de Redes de Distribución de Agua se ha modificado gradualmente hasta llegar a un nuevo enfoque al cual, en lo sucesivo en este documento, se hace referencia como enfoque moderno. En el enfoque moderno, el problema de diseño de redes de distribución de agua cambia, de tal forma que puede decirse que surge una nueva versión del problema. En este nuevo enfoque, el problema se basa en la optimización del costo del Diseño de Redes de Distribución de Agua pero ahora se centra en la solución de redes con topología mallada en lugar de topología ramificada. El problema con las redes ramificadas es la suspensión del servicio en distintos puntos de la red cuando ocurren roturas o fallas en alguna tubería, ya que por su naturaleza, las redes ramificadas cuentan únicamente con una única tubería para llegar de un punto a otro en la red. Ante esta problemática, el objetivo del problema de Diseño de Redes de Distribución de Agua ha cambiado considerablemente en este nuevo enfoque. El enfoque moderno[4] plantea el Diseño de Redes de Distribución de Agua malladas. Aquí, además de minimizar el costo del diseño de la red de distribución de agua, se pretende también que exista un mejor servicio para los usuarios. En estas redes, gracias a su estructura, es posible que el agua pueda llegar a un usuario de la red a través de diferentes caminos, ofreciendo mayores posibilidades de ofrecer al usuario de la red un servicio constante e ininterrumpido, aun en presencia de fallos en algunas tuberías de la red. A pesar de que el costo y la complejidad de implementar redes de distribución de agua malladas es mayor que cuando se implementan redes ramificadas, la implementación de redes malladas se justifica perfectamente porque dichas redes ofrecen mayor confiabilidad [Baños, 2006].

[4] Enfoque moderno es un término que se emplea en este trabajo para trabajos de la última década.

En el enfoque moderno el problema, de Diseño de Redes de Distribución de Agua, ha sido formulado generalmente mediante un modelo de programación no lineal, por los diferentes investigadores cuyos trabajos se muestran en la Tabla 2.2. En este modelo, las variables de decisión son los diámetros comerciales de las tuberías las cuales se definen como variables discretas en lugar de variables continuas. Al igual que en el enfoque clásico, se han propuesto diferentes métodos de solución para abordar el problema, generalmente basados en el uso de métodos de solución heurísticos (véase tabla 2.2).

Tabla 2.2 Problema RDA utilizando variables Discretas

Fecha	Autores	Método	Costo
1997	Savic y Walters	Algoritmo Genético	419000
1998a	Abebe et al.	Optimización Global	422000
1998b	Abebe et al.	Algoritmo Genético	424000
1998c	Abebe et al.	Búsqueda Aleatoria Controlada	439000
1999	Montesinos et al.	Algoritmo Genético	456000
1999	Cunha y Sousa	Recocido Simulado	419000
2001	Geem et al.	Búsqueda Armónica	419000
2003	Eusuff y Lansey	Algoritmo de la Rana que Salta	419000
2006	Reca et al.	Algoritmo Genético	419000
2010	**SEA-WDND**	Algoritmo Evolutivo	419000

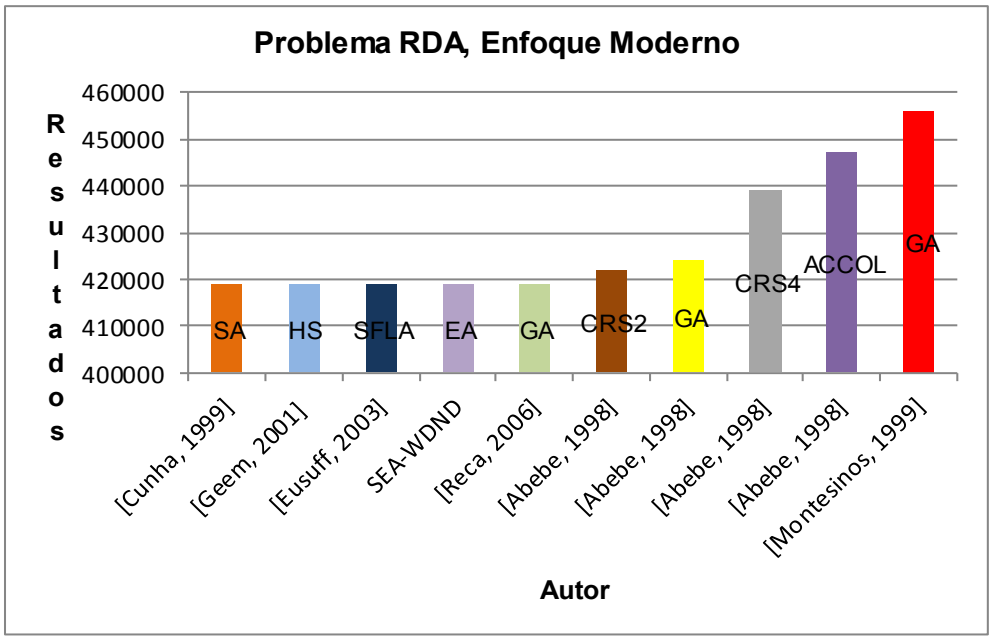


Figura 2.2 **Comparación de los costos utilizando Variables Discretas**

Los costos obtenidos con el enfoque de programación no lineal pueden verse de forma gráfica en la Figura 2.2. Puede observarse que diferentes investigadores coinciden en que la mejor solución del problema de Diseño de Redes de Distribución de Agua para la instancia Alperovits es 419,000 unidades y se encuentra codificada con un vector de valores flotantes. Estos resultados indican que el costo puede corresponder a la solución óptima del problema, aunque aún no existe la certeza debido a que las heurísticas no tienen prueba de optimalidad. Como puede observarse con la descripción anterior, el problema de Diseño de Redes de Distribución de Agua ha sido formulado y abordado desde dos enfoques diferentes (clásico y moderno) obteniendo resultados distintos. Puede decirse que, tanto el modelo matemático para representar el problema como las técnicas de solución aplicadas para resolverlo varían de forma significativa. Con esto, puede concluirse que el enfoque clásico y el enfoque moderno no son completamente comparables porque son estudios con diferentes formulaciones y condiciones distintas del problema de Diseño de Redes de Distribución de Agua.

Los métodos de solución propuestos para abordar el problema de Diseño de Redes de Distribución de Agua, son generalmente técnicas heurísticas. Estas se han utilizado para trabajos con características similares a las del presente problema. Con frecuencia, se han utilizado a los algoritmos evolutivos para abordar el problema, permitiendo obtener muy buenos resultados. También se han utilizado otras heurísticas para la solución del problema de Diseño de Redes de Distribución de Agua, tal como Colonias de Hormigas [Gil, 2011].

El desarrollo de este trabajo doctoral se encuentra dentro del enfoque moderno, en el cual el objetivo es minimizar el costo del diseño de la red de distribución en redes de distribución de agua malladas, siendo los diámetros comerciales de las tuberías las variables de decisión manejadas como variables discretas. El modelo matemático que representa al problema de Diseño de Redes de Distribución de Agua, ha sido formulado en este trabajo mediante dos modelos matemáticos independientes que deben resolverse, para obtener una solución para el problema: el modelo de programación lineal y el modelo de satisfacción de restricciones.

2.2. Algoritmos Evolutivos para el problema de Diseño de Redes de Distribución de Agua

Los Algoritmos Evolutivos son heurísticas que se encuentran dentro del paradigma de la computación evolutiva. Se pueden definir como métodos computacionales que simulan los procesos definidos en la teoría de la evolución de los seres vivos [Holland, 1975].

Los primeros trabajos científicos sobre algoritmos evolutivos surgen a finales de los años cincuenta, cuando diferentes investigadores trabajaban en paralelo, de forma independiente, diseñando algoritmos evolutivos que se basaban en la teoría de la creación y evolución de las especies presentada por Darwin [Darwin, 1859]. Las investigaciones realizadas desde los años cincuenta permitieron que los algoritmos evolutivos puedan ser clasificados, en la actualidad, con diferentes enfoques: programación evolutiva [Fogel, 1965], estrategias de evolución [Rechenberg 1973], algoritmos genéticos [Holland, 1975], e incluso la programación genética [Koza, 1992], la cual es considerada como otra variante de los algoritmos evolutivos, derivada de los algoritmos genéticos. Estas estrategias se basan en los principios de la evolución natural, presentando así un esquema de funcionamiento común y elementos en los que coinciden. Sin embargo, cada una de ellas incorpora sus propios detalles de diseño e implementación en la codificación de las soluciones, los tipos de mutación, recombinación y selección. A pesar de que los enfoques de algoritmos evolutivos se originaron de forma independiente y con motivaciones diferentes, en la actualidad, es cada vez más difícil distinguir las diferencias reales entre los tipos de algoritmos evolutivos existentes; para una descripción más detallada puede consultarse [Alba, 2011].

En [Duarte, 2006] se muestran diferentes publicaciones en las que se proponen Algoritmos Evolutivos para abordar problemas combinatorios. Este trabajo se enfoca concretamente en la estrategia de los algoritmos evolutivos utilizando una codificación de las soluciones real y haciendo énfasis en el operador de cruce frente al operador de mutación, por ser el mecanismo principal de búsqueda.

La teoría de los algoritmos genéticos fue propuesta por [Holland, 1975] y desarrollada posteriormente por [Goldberg, 1989], con la finalidad de representar computacionalmente los mecanismos empleados por la naturaleza para la evolución de las especies. Esta teoría explica los cambios evolutivos en una población resaltando la supervivencia y la reproducción de las especies. Así mismo, en ella se expresa que los individuos de una población producen más descendencia de la necesaria, ocasionando con ello un crecimiento exponencial de la población. No obstante, existen factores ambientales que limitan y regulan el crecimiento de la población, por ejemplo el hecho de que los individuos compiten entre ellos para obtener comida y otros recursos [Amor, 2008].

Los algoritmos genéticos son el tipo de algoritmos evolutivos más comúnmente utilizado. Son técnicas estocásticas en las que los métodos de búsqueda imitan fenómenos de la evolución natural de los seres vivos. Estos algoritmos imitan el comportamiento natural de los individuos y de las poblaciones. Así, los individuos nacen, crecen, se reproducen y mueren. Pero además, en la mayoría de las especies se observa la supervivencia del individuo más apto o mejor preparado. La simulación de los procesos evolutivos ha sido implementada mediante técnicas computacionales, con el objetivo de dar solución a problemas de optimización de diversa índole. Sobre algoritmos genéticos existen diversas publicaciones en la literatura [Savic, 1997; Abebe, 1998b; Montesinos, 1999; Amor, 2005; Reca, 2006].

A cada uno de los procesos definidos en la teoría de la evolución le corresponde un proceso análogo que es implementado en los algoritmos computacionales. Puede decirse que un individuo es representado mediante una solución computacional. Asimismo, una población es representada mediante un conjunto de soluciones computacionales. Básicamente, son estos los dos elementos primordiales de un algoritmo evolutivo. La población de individuos evoluciona a través de numerosas generaciones en las cuales los individuos son seleccionados, mutados y combinados para producir descendencia.

Los Algoritmos Evolutivos simulan la existencia de generaciones mediante un proceso iterativo. En cada generación pueden existir diversos individuos, tal como ocurre en la selección natural de las especies; gracias a las diferentes operaciones que se

realizan entre ellos la población evoluciona a través del tiempo, dando como resultado individuos que posiblemente son la solución de un problema. Cada individuo, de la población, está constituido por un conjunto de valores que describen a una solución específica. Cada solución está codificada en cromosomas, los cuales son cadenas de caracteres que representan una analogía con las cadenas de caracteres encontradas en el DNA. Los caracteres de una solución computacional pueden ser alfabéticos, numéricos, binarios, entre otros. Por otra parte, el proceso de evolución se logra con la reproducción de los individuos gracias a la existencia de los operadores clásicos de selección, cruzamiento (o combinación) y mutación.

La selección consiste en definir, basándose en una función de evaluación, a los individuos aptos para convertirse en padres. Es decir, una vez que se evalúa la aptitud, la selección determina cuáles individuos pueden sobrevivir en las siguientes generaciones, a través de sus descendientes. Se dice que entre mejor sea la calidad del individuo, existen mayores probabilidades de que éste sea seleccionado en el proceso evolutivo para pasar su material genético a nuevos individuos y así continuar viviendo en las siguientes generaciones. El proceso de reproducción consiste precisamente en generar nuevos individuos a partir de la combinación de dos individuos padres, los cuales heredan a los descendientes, algunas de sus características.

La generación de nuevos individuos se obtiene después de aplicar el operador llamado cruce y es un proceso en el que se combinan cromosomas de los padres para generar nuevos individuos, con características comunes a ambos padres. La mutación es un proceso que puede ocurrir a cualquier individuo y consiste en modificar parte de su material genético. Con esto puede decirse que se “genera” un nuevo individuo, ya que algunos de sus genes son modificados de acuerdo con una función de probabilidad. Generalmente, no existe una regla que indique cuantos genes se deben modificar, tal como ocurre en la teoría de la evolución natural de los seres vivos. Como puede observarse, el proceso de evolución se rige mediante los operadores de cruce y de mutación, los cuales permiten generar nuevos individuos en una población. Después de aplicar los operadores se observa una fase de remplazo que consiste en sustituir la población actual, o parte de ella, por otros individuos que continúan la evolución de la población. Los algoritmos evolutivos presentan algunas ventajas por las cuales pueden elegirse para solucionar una amplia variedad de problemas de optimización. Entre las

ventajas que presentan los Algoritmos Evolutivos, las más importantes se refieren a que son algoritmos conceptualmente simples y fáciles de entender. Además, los Algoritmos Evolutivos pueden aprovechar el uso de entornos de cómputo paralelo para obtener soluciones de calidad en un tiempo relativamente corto. Sin embargo, un algoritmo evolutivo es un área de estudio muy extensa por la cantidad de componentes, parámetros y métodos de operación que utiliza para implementar la evolución de una población. El comportamiento del algoritmo es único y especial de acuerdo con la definición de parámetros y definición de estrategias que utiliza, tal como se muestra en el apartado 5.2 de estudios experimentales. Entre las desventajas más importantes de los algoritmos evolutivos, es que éstos son considerados como mecanismos de búsqueda ciegos, ya que únicamente se basan en la función de aptitud para proseguir con la búsqueda de las mejores soluciones, para problemas de optimización.

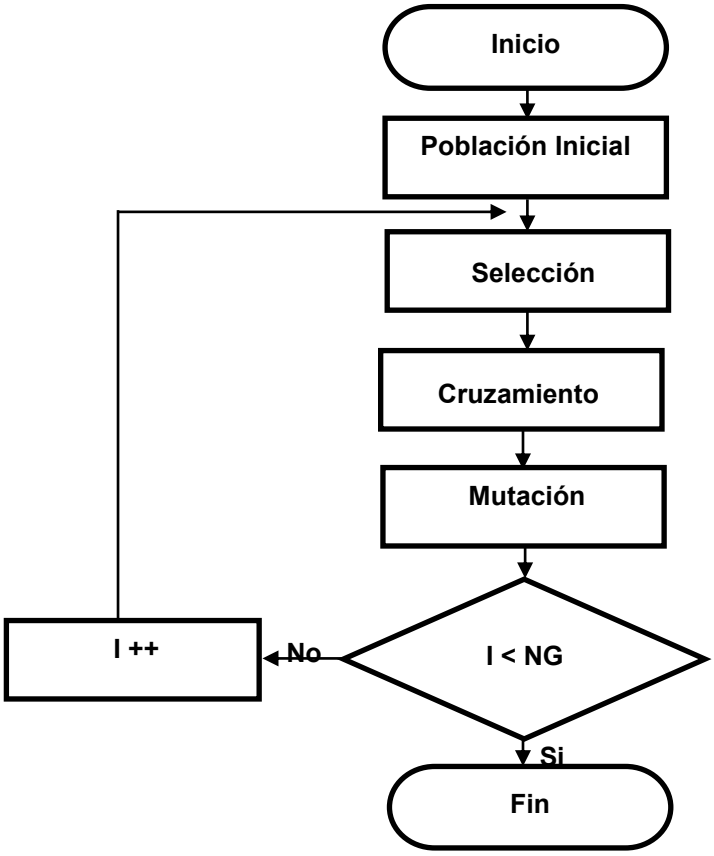


Figura 2.3 Esquema básico de un Algoritmo Evolutivo

La Figura 2.3 muestra el funcionamiento básico de un algoritmo evolutivo. La nomenclatura utilizada en los algoritmos evolutivos está estrechamente relacionada con su inspiración biológica. En este sentido, es común utilizar el término “población de individuos” para definir un conjunto de soluciones para un problema dado. También se define el término cromosoma, por ser una analogía con las cadenas de caracteres encontradas en el DNA, como una codificación compuesta por unidades llamadas genes. Puede decirse que un cromosoma es una estructura de datos que representa una posible solución al problema.

En los algoritmos genéticos tradicionales, la codificación de las soluciones puede hacerse mediante una representación alfabética o binaria. La representación de una solución se refiere a los tipos de variables del problema y se conoce como fenotipo; pueden ser valores binarios, enteros, reales, entre otros. La representación del individuo es llamada genotipo y está constituida por cadenas de bits conocidas como cromosomas, las cuales son estructuras que representan una solución potencial a un problema específico. La representación constituye una correspondencia entre las soluciones (fenotipo) y la codificación de las variables (genotipo). Originalmente, la representación de un algoritmo se hacía mediante cadenas binarias. Sin embargo, esta representación era una limitación importante de estos algoritmos. Recientemente, esta limitación ha sido eliminada, en algoritmos evolutivos utilizando en su lugar valores enteros y valores reales [Reca, 2006]. Desde el punto de vista computacional, un algoritmo evolutivo contiene elementos como una población inicial, representación, función de evaluación, operadores genéticos, entre otros [Duarte, 2006]. La población inicial es un conjunto de posibles soluciones, comúnmente generadas de forma aleatoria, para un problema dado, cada solución recibe el nombre de individuo.

Por otra parte, la función de evaluación consiste en determinar la calidad de los individuos de la población con base en la función objetivo del problema. Si el problema es de minimizar, la función de evaluación determinará que las soluciones de mejor calidad serán aquellas que representen los costos más bajos. Por lo contrario, si el problema es de maximizar la función de evaluación determinará que las soluciones de mejor calidad serán aquellas soluciones que presenten los costos más altos.

Los operadores que se utilizan, de manera habitual, en un algoritmo evolutivo son métodos probabilísticos que permiten obtener nuevos individuos mediante la modificación estocástica de los antepasados, utilizando para ello los operadores de selección cruce y mutación. El operador de selección consiste en determinar cuáles individuos son aptos para convertirse en padres y transmitir a los descendientes al menos parte de su contenido genético. En este proceso se pueden elegir con una mayor probabilidad a los individuos que presenten un valor más elevado de la función de evaluación. Sin embargo, existen diferentes criterios para seleccionar a los padres. En la literatura existen revisiones detalladas de los diferentes métodos de selección que pueden utilizarse [Reeves, 2003]. El operador de cruce es una de las aportaciones fundamentales de los algoritmos evolutivos. Consiste en combinar genes de dos padres para generar uno o más hijos. En la literatura, se pueden encontrar diferentes métodos de cruce, que indican la forma en que dos cromosomas pueden combinarse para generar descendientes.

La mutación consiste en realizar una modificación en cierto número de genes de un individuo, determinado por el operador de mutación. Al operador de mutación, generalmente, se le asignan valores que determinan cambiar de 1 a 10% de los genes de un individuo. La mutación puede ser de diferentes tipos [Michalewicz, 1992] y se emplea como un mecanismo para preservar la diversidad de los individuos intentando explorar exhaustivamente zonas del espacio de búsqueda.

Actualmente, existe una gran variedad de problemas teóricos y prácticos que son resueltos mediante el uso de algoritmos evolutivos. En la literatura, destacan trabajos con AE para problemas de calendarización de horarios (Job Shop Scheeduling Problem) [Cruz, 2010a], problemas de transporte (Vehicle Routing Problem) [Cruz, 2012], problemas del Transporte con Ventanas de Tiempo [Rodriguez, 2010], problemas de Diseño de Redes de Distribución de Agua (Water Distribution Network Design) [Cruz, 2010c], entre otros.

Para abordar el problema de Diseño de Redes de Distribución de Agua, se han desarrollado múltiples trabajos. Algunos de ellos son considerados como clásicos [Alperovits, 1977]. También existen trabajos recientes para nuevas versiones del problema de Diseño de Redes de Distribución de Agua [Reca, 2008].

El número de publicaciones, para el problema de Diseño de Redes de Distribución de Agua, es realmente extenso si se consideran todos los trabajos presentados en la literatura. Sin embargo, en este trabajo se analizan únicamente las principales publicaciones que tienen relación directa con la técnica de solución propuesta en esta tesis, tabla 2.3.

Tabla 2.3 Algoritmos Evolutivos para el Problema RDA

Fecha	Autores	Costo
1997	Savic y Walters	419000
1998	Abebe et al.	424000
1999	Montesinos et al.	456000
2003	Matías	419000
2006	Reca et al.	419000

Los algoritmos evolutivos han sido causado interés en la comunidad científica y han sido implementados por varios investigadores. Se han propuesto modificaciones [Montesinos, 1999] y mejoras a estos algoritmos [Dandy, 1996; Walters, 1999]. Otros estudios han demostrado la eficacia de los algoritmos evolutivos en la optimización de sistemas de distribución de agua [Nazif, 2009]. Basándose en los resultados presentados en la literatura, es posible decir que estas técnicas son una buena alternativa para dar solución al problema RDA, ya que combinan elementos de búsqueda conducida con elementos de búsqueda estocástica. Adicionalmente, trabajan con una búsqueda multidireccional mediante poblaciones de individuos que representan soluciones potenciales para el problema. Finalmente, los algoritmos evolutivos tienen características que permiten que sean relativamente fáciles de llevar a ambientes de cómputo paralelo.

En la literatura existen numerosas publicaciones en las cuales se presentan trabajos muy detallados de heurísticas paralelas aplicadas a diferentes problemas [Alba, 2005]. Puede observarse que las aplicaciones paralelas obtienen resultados más eficientes, y en algunos casos más eficaces, que los resultados obtenidos por aplicaciones secuenciales [Alba, 2002]. Es importante mencionar que existen trabajos para resolver el problema de Diseño de Redes de Distribución de Agua utilizando estrategias paralelas mediante cómputo paralelo en clúster de computadoras [Baños, 2006]. No obstante, no se encontraron publicaciones en la literatura de trabajos donde se utilicen técnicas de

paralelismo en ambiente Grid para resolver el problema de Diseño de Redes de Distribución de Agua. De acuerdo con esto, puede decirse que el algoritmo evolutivo propuesto en este trabajo doctoral, representa un trabajo innovador para la solución del problema de Diseño de Redes de Distribución de Agua.

Capítulo 3. Fundamentos del Problema

En este capítulo, se presenta una descripción teórica del problema de Diseño de Redes de Distribución de Agua. También se describen las instancias de prueba que se resuelven en este trabajo doctoral. Finalmente, en este capítulo, se presenta la formulación matemática definida en esta tesis para abordar el problema de Diseño de Redes de Distribución de Agua.

3.1. Descripción del Problema de Diseño de Redes de Distribución de Agua

El problema de Diseño de Redes de Distribución de Agua es un problema de gran interés para diversos investigadores, dentro de la optimización combinatoria, por su gran aplicación práctica. Durante más de tres décadas el problema de Diseño de Redes de Distribución de Agua, ha sido estudiado ampliamente. El problema, consiste en determinar los componentes que formarán parte de la red de tal forma que la red resultante sea una red de costo mínimo que permita brindar un servicio adecuado a los usuarios de acuerdo con sus necesidades, considerando que éstas pueden variar dependiendo de los diferentes usuarios de la red. Para brindar mayor confiabilidad en el servicio a los usuarios de la red, sobre todo en situaciones en las que el servicio de agua es imprescindible, generalmente se implementan redes malladas, Figura 3.1. La red de distribución de agua puede estar formada por diferentes componentes interconectados entre sí. Entre los componentes más comunes, para el diseño de la red, se encuentran tuberías, bombas, fuentes de abastecimiento, entre otros. Las tuberías comerciales, que están disponibles en diferentes diámetros y materiales, permiten llevar el agua desde las fuentes hasta los usuarios de la red. Las bombas de potencia son elementos indispensables cuando la técnica de distribución del agua es bombeo. Finalmente, las fuentes de abastecimiento, son los puntos de suministro principales de los cuales se

extraen los recursos hídricos, pueden ser ríos, arroyos, manantiales, pozos, entre otros. Éstas, son imprescindibles para distribuir el agua a los usuarios de la red.

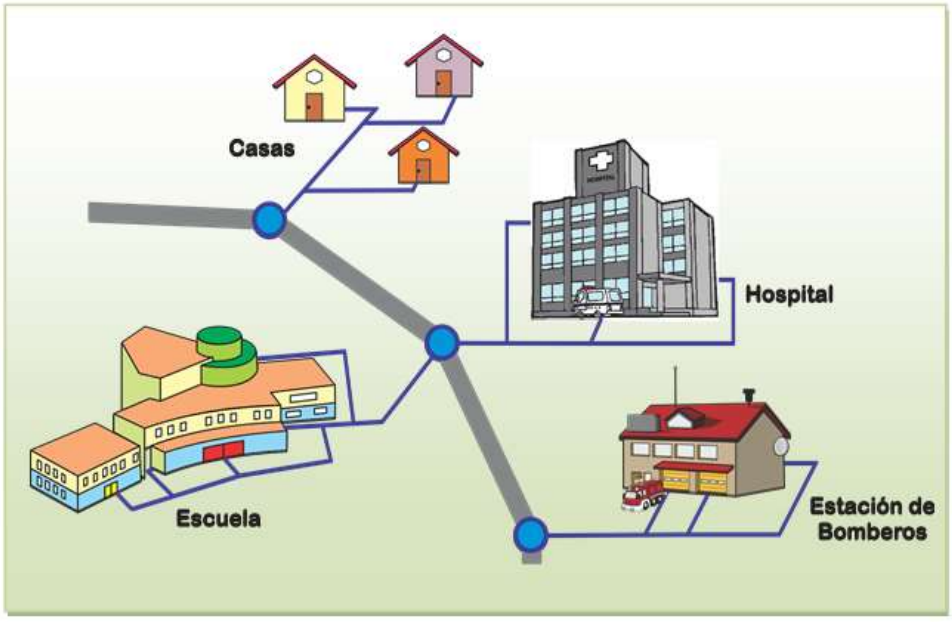


Figura 3.1 Componentes de una Red

El problema de redes de distribución de agua puede estudiarse desde diferentes etapas, como son diseño, mantenimiento, distribución del agua, entre otras. Este trabajo doctoral se enfoca en la fase de diseño. La fase de diseño de una red de distribución de agua es muy importante porque de ésta depende, en gran parte, el buen funcionamiento de la misma. Previo a la fase de diseño, se debe hacer un estudio de campo para recopilar información del área de estudio; se debe definir una topología, de las descritas en I.1, para representar a la red. También es necesario definir la ubicación física de los componentes de la red, de los usuarios y de las fuentes de abastecimiento, así como el nivel del agua de los depósitos o la altura en la que se encuentran las estaciones de bombeo, en caso de que existan. Adicionalmente, es necesario identificar el gasto hídrico de los usuarios, los costos de operación de la red, entre otros. En el diseño de la red es importante considerar las condiciones topográficas de la red, las características de los componentes, el número de usuarios de la red, las demandas hídricas de los usuarios, entre otros. La elección de los diámetros de las tuberías, es el último paso para el diseño de una red de distribución de agua. Antes de realizar este paso el diseñador de la red

debe elegir la topología, los componentes, las fuentes de abastecimiento disponibles y las demandas hídricas de los usuarios y plasmarlas en un diagrama. Una vez definido el diagrama, el objetivo de realizar una planeación de las redes de distribución de agua es minimizar el costo del diseño de la red. Es decir, se deben encontrar los componentes de la red que permitan que ésta tenga el menor costo posible considerando también que, con las características y componentes definidos, la red pueda brindar a los usuarios un suministro de agua adecuado.

Las redes de distribución de agua pueden representarse de forma gráfica utilizando la teoría de grafos. [Ostfeld, 2005; Berge, 1973; Chistofides, 1975; Bondy, 1976; Sedgewick, 1984; Diestel, 2010] muestran información detallada de la teoría de grafos. A continuación se presentan de manera general algunos conceptos importantes y propiedades de los grafos utilizadas para el desarrollo del Algoritmo Evolutivo para el problema abordado en esta tesis.

Un grafo es una representación gráfica que permite modelar problemas de conectividad [Sedgewick, 1984]. Los grafos son una estructura de datos no lineal que puede utilizarse para modelar y solucionar gran número de problemas reales, como son redes de telecomunicaciones, redes de transportes, redes hidráulicas, entre otras. Un grafo G se define como un conjunto de vértices o nodos *V* y un conjunto de arcos *A*, cada uno de ellos se encarga de unir un vértice con otro. Los arcos que unen a los vértices de un grafo se llaman también aristas y se representan mediante un par de elementos (v_i,v_j), dónde generalmente i es diferente de j y ambos son vértices del grafo. Si en un grafo los arcos tienen dirección, el grafo se llama grafo dirigido. En caso contrario se habla de grafos no dirigidos. En un grafo dirigido G = (V,E), V es el conjunto de vértices $V = \{v_1, v_2, v_3, ..., v_n\}$ y E es un conjunto de pares ordenados llamados arcos $E = \{v_i, v_j\}$. La representación de un arco es mediante los elementos (v_i,v_j) que indican la conexión de un nodo origen (i) con un nodo destino (j). En un grafo no dirigido G = (V,E), V es el conjunto de vértices $V = \{v_1, v_2, v_3, ..., v_n\}$ y E es un conjunto de pares no ordenados llamados arcos $E = \{v_i, v_j\}$. El total de arcos y de vértices en un grafo se denotan mediante |E| y |V| respectivamente y con éstos es posible imaginar las dimensiones de una red. En resumen, se dice que en un grafo los vértices representan los objetos y los

arcos representan las relaciones entre esos objetos. De esta forma se observa que los vértices y los arcos de un grafo son conceptos abstractos que pueden corresponder a objetos concretos cuando se aplica a problemas reales. Para representar el problema de redes de distribución de agua, abordado en esta tesis, puede utilizarse un grafo no dirigido como el que muestra la Figura 3.2. Este grafo, que representa a la red Alperovits [Alperovits, 1977], está compuesto por 7 vértices y 8 arcos; los vértices representan a las fuentes de abastecimiento y a los usuarios finales de la red que pueden ser usuarios de zonas urbanas, zonas industriales, zonas de riego, entre otras. Los arcos en el grafo representan a los elementos de conexión y pueden ser tuberías, bombas, entre otros. La estructura de la red modelada mediante el grafo puede representarse computacionalmente mediante el uso de listas. El etiquetamiento de los nodos de un grafo consiste en asignar de manera única un nombre para cada uno de los vértices del grafo. Así mismo se debe asignar también un nombre único a cada uno de los arcos del grafo. Así en la Figura 3.2 los vértices del grafo tienen etiquetas entre corchetes mientras que los arcos están etiquetados mediante números del 1 al 8.

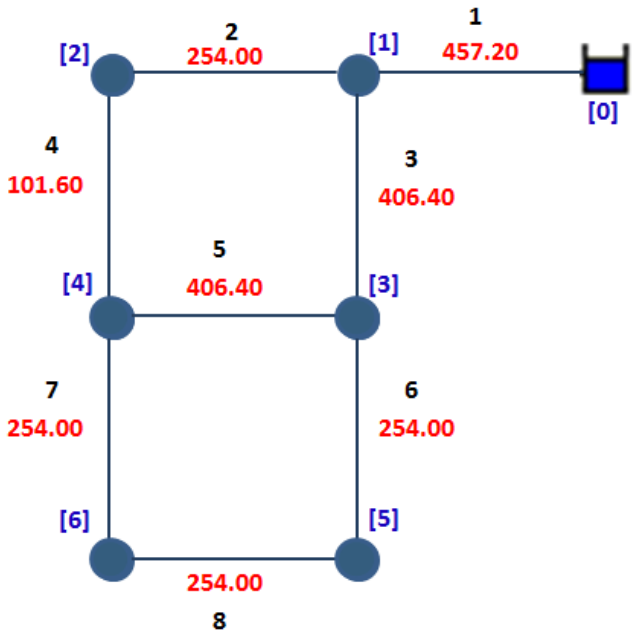


Figura 3.2 Configuración inicial de la Red Alperovits (Adaptada de [Alperovits, 1977])

En un grafo es posible hablar de trayectorias o caminos. En la Figura 3.2, se observa que una trayectoria en un grafo es una secuencia de arcos por ejemplo, los arcos

1,2,4,5,6 forman una trayectoria que permitiría llevar agua desde el depósito [0] hasta el usuario [5] de la red. El número de arcos que inciden en un vértice indica precisamente el grado de un vértice. Por ejemplo el vértice [4] de la Figura 3.2 tiene grado 3. De forma implícita se observa que cada arco de la trayectoria inicia justo donde termina el arco anterior; cuando dos vértices están conectados directamente por un arco se dice que éstos son adyacentes. La conectividad de un grafo no dirigido se refiere a una trayectoria entre dos vértices cualesquiera de un grafo. La conectividad de un grafo es un concepto básico en la teoría de grafos que además sirve como métrica para la confiabilidad y tolerancia a fallos en una red.

El problema de Diseño de Redes de Distribución de Agua puede abordarse con diferentes enfoques. Así mismo, el problema puede tener distintos parámetros dependiendo de la topología de la red con la que se trabaje, las demandas de los usuarios de la red y los componentes que constituyan la red. Así mismo, influye también la técnica utilizada para la distribución del agua. Sin embargo, de manera general, el problema consiste en dimensionar las aristas del grafo mediante diámetros de tuberías comerciales para formar una red de costo mínimo que permita satisfacer las demandas hídricas de los usuarios mediante presiones mínimas y máximas requeridas para ofrecer un buen servicio. Puede decirse que el problema de Diseño de Redes de Distribución de Agua consiste en encontrar el costo mínimo de la red y al mismo tiempo ofrecer un servicio adecuado a los usuarios. Cuando la red se dimensiona con los diámetros más pequeños, de la lista de diámetros comerciales disponibles, se obtiene una red de costo mínimo. No obstante, puede ocurrir que algunos usuarios de la red tengan un servicio deficiente con presiones inferiores a las presiones mínimas requeridas. Por el contrario, cuando la red se dimensiona con los diámetros comerciales más grandes, de la lista de diámetros comerciales disponibles, generalmente todos los usuarios de la red pueden tener un servicio de calidad en el que las presiones del servicio que reciben satisfacen sus necesidades. Sin embargo, bajo estas condiciones el costo del diseño de la red es muy elevado, por lo que no se logra el objetivo de tener una red de costo mínimo. En este punto se observa que en realidad se busca cumplir con dos objetivos básicos para el problema. El primer objetivo es tener una red de costo mínimo y el segundo objetivo es obtener al menos las presiones mínimas requeridas para ofrecer al usuario un servicio adecuado a sus necesidades. En el caso de la red Alperovits al dimensionar la red con el

mayor diámetro que es de 609.6 mm, las presiones de la red para el servicio a los usuarios son mayores a 40 mca[4] pero el costo de la red es de 4,400,000. En caso de dimensionar la red con el menor de los diámetros, que es de 25.4 mm, el costo de la red es de 16,000 pero las presiones que obtienen las usuarios con el servicio son negativas, lo cual significa que no hay presiones. En hidráulica una presión negativa es una presión menor a la presión atmosférica. Con presiones atmosféricas de -1 se habla del vacío, de que hay presión cero, pero matemáticamente Epanet calcula la presión y obtiene valores negativos que indican grandes pérdidas de carga por causa de presiones elevadas en el diámetro de tubería definido, al intentar cumplir con caudales que se requieren para dar servicio adecuado a los usuarios.

Como puede observarse, con el párrafo anterior, existen casos en los que se logra tener una red de costo mínimo pero el servicio que brinda a los usuarios es deficiente. En otros casos, los usuarios pueden tener un servicio de calidad, acorde a sus necesidades, pero con una red de costo elevado. Sin embargo, es extremadamente complejo obtener una red de costo mínimo que permita a los usuarios tener un servicio de calidad [Baños, 2006]. En este trabajo primero se asegura que los usuarios de la red tengan un servicio que cumpla sus necesidades hídricas y posteriormente se trabaja en encontrar el diseño de la red de distribución de agua con el menor costo posible. En concreto, el problema de Diseño de Redes de Distribución de Agua consiste en:

1) Determinar los diámetros de tuberías para obtener una red de costo mínimo.
2) Asegurar que todos y cada uno de los usuarios de la red puedan tener las presiones mínimas definidas para considerar que tienen un buen suministro de los recursos hídricos.
3) Asegurar que se cumplan las restricciones de la conservación de la masa y la energía para que la red tenga un funcionamiento adecuado, evitando rupturas en las tuberías u otros comportamientos no deseados.

[4] mca se refiere, por sus siglas metros por columna de agua, se refiere a la presión que se necesita para elevar el agua un metro.

Para concluir con la explicación teórica del problema, a continuación se muestran algunos posibles escenarios que ilustran el problema de Diseño de Redes de Distribución de Agua, utilizando para ello la instancia de prueba *Two-Looped Network* [Alperovits, 1977], la cual se considera como una instancia pequeña, pero a la vez muy útil para ejemplificar y realizar pruebas con facilidad y precisión. La red *Two-Looped*, que en lo sucesivo se hace referencia a ella como red Alperovits, es una red mallada simple de 2 lazos, 7 nodos y 8 tuberías, Figura 3.2. Las etiquetas entre corchetes representan a los nodos, los cuales pueden ser depósitos o nodos de consumo. Las tuberías están etiquetadas mediante números del 1 al 8, para esta instancia de prueba. Asimismo, los pesos que se encuentran representados con valores reales, justo debajo de las tuberías, representan diámetros de tuberías elegidas de una lista de diámetros comerciales y son precisamente las variables de decisión del problema abordado. La red tiene una sola presa con una elevación de 210 metros (etiqueta [0]), de la cual se obtiene el agua. Para el diseño de la red Alperovits se deben configurar 8 diámetros de tuberías, de la lista de diámetros comerciales disponibles, véase tabla 1 del Apéndice I. El número de posibles configuraciones es para esta instancia es de $14^8 = 1.4758 * 10^9$; la longitud de cada tubería de la red es de 1000 m. Para cumplir las demandas de los usuarios es importante que en cada nodo existan presiones mínimas de 30 mca y máximas de 100 mca.

Escenario 1. Configuración Inicial de la Red

Para cada uno de los siguientes escenarios se ha definido una configuración inicial. Es decir, a cada una de las tuberías de la red se les ha asignado un diámetro de la lista de diámetros comerciales, Figura 3.2. Una vez definida la configuración, se ha utilizado el evaluador hidráulico Epanet para saber cuáles serán las presiones resultantes (Tabla 3.1) que los usuarios obtendrán con el diseño definido en la Figura 3.2. Como puede observarse, en la Tabla 3.1, los nodos 5 y 6 no alcanzan las presiones mínimas requeridas, puesto que tienen presiones de 27.84 y 29.60 respectivamente. De acuerdo con los resultados, la configuración cuyo costo seria 449 000 unidades, se rechaza porque algunos de los usuarios de la red tendrían presiones menores a las requeridas.

Tabla 3.1 Presiones de una Red

Tubería	Conexión	Diámetro	Costo	Nodo	Presión
1	0-1	457.20	130.00	0	Depósito
2	1-2	254.00	32.00	1	53.25
3	1-3	406.40	90.00	2	32.67
4	2-4	101.60	11.00	3	43.02
5	3-4	406.40	90.00	4	37.61
6	3-5	254.00	32.00	5	27.84
7	4-6	254.00	32.00	6	29.60
8	5-6	254.00	32.00		

Escenario 2. Aumentar diámetro de tubería 6

Con el diseño resultante en el escenario 1 los nodos 5 y 6 no alcanzan las presiones mínimas requeridas, entonces una opción es intentar ajustar las presiones rediseñando la red de distribución de agua. Para lograr el objetivo de obtener la presión mínima requerida en el nodo 5 podrían modificarse la tubería 6 o bien la tubería 8. En este escenario se cambia la tubería 6 de diámetro 254.00 por una tubería de diámetro 304.8, Figura 3.3.

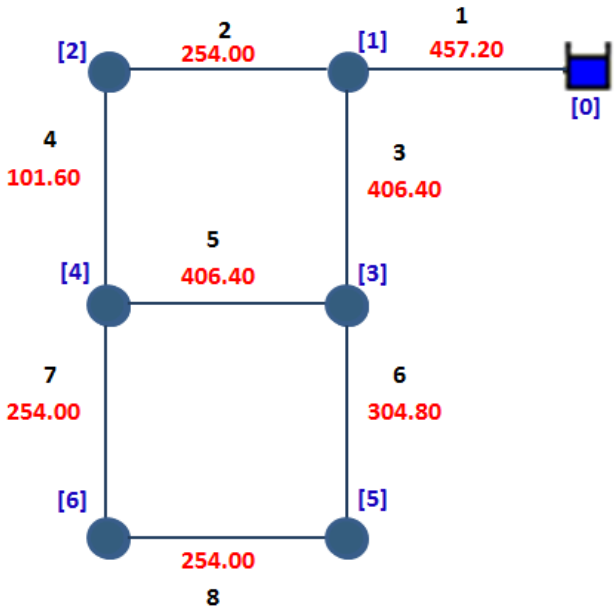


Figura 3.3 Configuración cambiando tubería 6

Al aumentar la tubería 6 se observa la presión, en el nodo 5, aumenta de 27.84 a 30.62. Sin embargo, puede observarse que, en este escenario, el nodo 6 aún no alcanza la presión requerida. Incluso, al cambiar la tubería 6 por una de diámetro mayor, ahora la presión en el nodo 6 es menor. Con la tubería 6 de diámetro 254.00 la presión en el nodo 6 era 29.60 (escenario 1) y ahora con la tubería de diámetro 304.80 es 28.99, tabla 3.2. En este escenario el costo total de la red es de 467,000.

Tabla 3.2 Presiones cambiando tubería 6

Tubería	Diámetro	Costo	Nodo	Presión
1	457.20	130.00	0	Depósito
2	254.00	32.00	1	53.25
3	406.40	90.00	2	34.06
4	101.60	11.00	3	42.71
5	406.40	90.00	4	40.01
6	304.80	50.00	5	30.62
7	254.00	32.00	6	28.99
8	254.00	32.00		

Escenario 2. Aumentar diámetro de tubería 8

Al cambiar la tubería 8 de diámetro 254.00 por una tubería de diámetro 304.80, Figura 3.4, no se alcanza la presión en el nodo 5,Tabla 3.3.

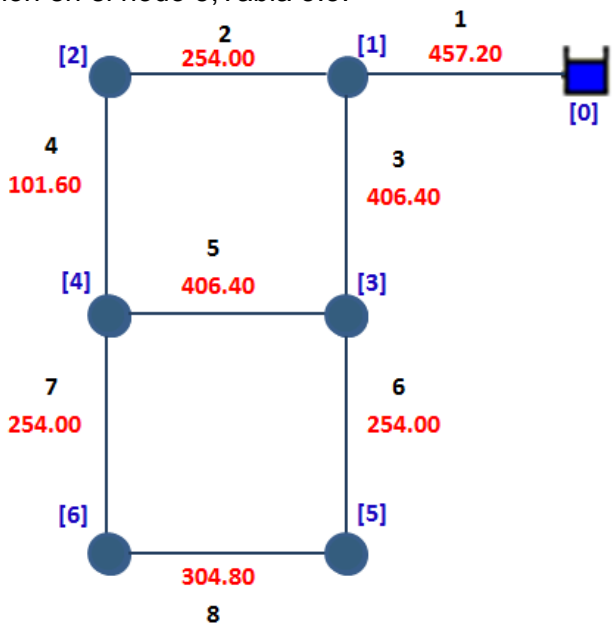


Figura 3.4 Configuración cambiando tubería 8

Tabla 3.3 Presiones cambiando tubería 8

Tubería	Diámetro	Costo	Nodo	Presión
1	457.20	130.00	0	Depósito
2	254.00	32.00	1	53.25
3	406.40	90.00	2	31.46
4	101.60	11.00	3	43.26
5	406.40	90.00	4	39.02
6	254.00	32.00	5	29.11
7	254.00	32.00	6	30.02
8	304.80	50.00		

Escenario 3. Disminuir diámetro de tubería 8

Otro posible escenario es ajustar la presión en los nodos, rediseñando la red mediante disminución en las tuberías. Al cambiar el diámetro de la tubería 8 de diámetro 254.00 por una tubería de diámetro 203.20, Figura 3.5, las presiones se acercan a la presión mínima requerida.

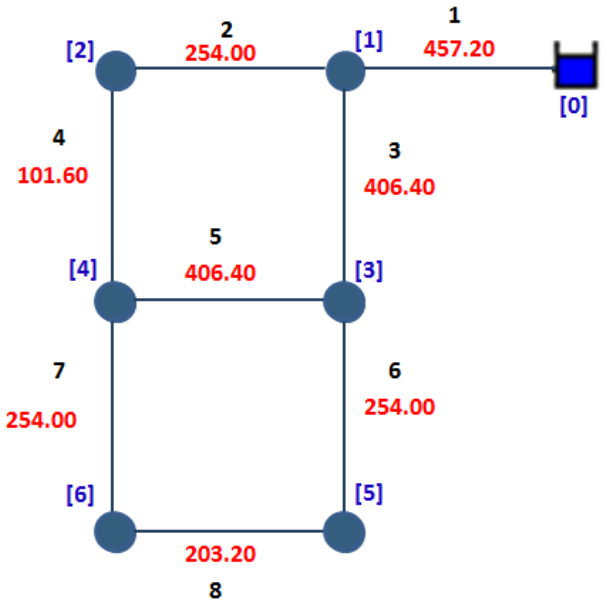


Figura 3.5 Configuración disminuyendo diámetro de tubería 8

Puede observarse que en el nodo 5 la presión cambia de 27.84 a 28.01 y en el nodo 6 cambia de 29.60 a 29.65. Con este escenario las presiones en todos los nodos de la red se acercan a las presiones mínimas requeridas, tabla 3.4.

Tabla 3.4 Presiones disminuyendo tubería 8

Tubería	Diámetro	Costo	Nodo	Presión
1	457.20	130.00	0	Depósito
2	254.00	32.00	1	53.25
3	406.40	90.00	2	32.54
4	101.60	11.00	3	43.04
5	406.40	90.00	4	37.39
6	254.00	32.00	5	28.01
7	254.00	32.00	6	29.65
8	203.20	23.00		

Escenario 4. Cambiar diámetro de tubería 8 por el menor de los diámetros

Al cambiar la tubería 8 de por el menor de los diámetros disponibles (25.40), Figura 3.6, se observa que todos y cada uno de los nodos de la red alcanzan las presiones mínimas requeridas, tabla 3.5. La configuración de la Figura 3.6 podría ser la solución óptima para la instancia Alperovits, ya que se alcanzan las presiones mínimas requeridas para todos los nodos de la red y se minimiza también el costo del diseño de la red (419,000 unidades).

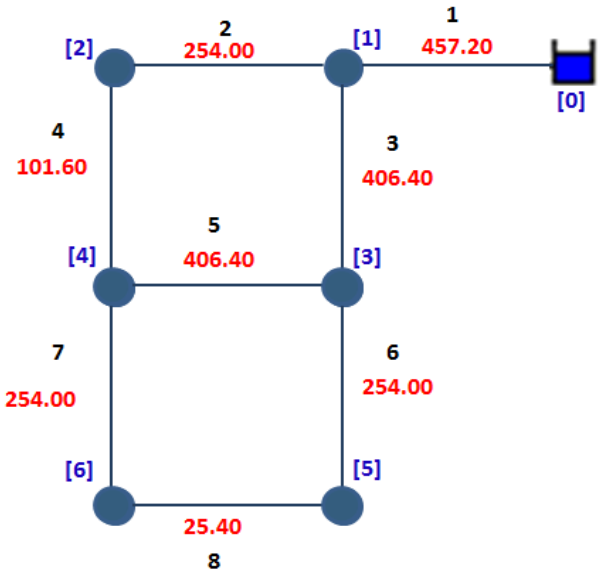


Figura 3.6 Configuración con menor diámetro en tubería 8

Tabla 3.5 Presiones con menor diámetro en tubería 8

Tubería	Diámetro	Costo	Nodo	Presión
1	457.20	130.00	0	Depósito
2	254.00	32.00	1	53.25
3	406.40	90.00	2	30.47
4	101.60	11.00	3	43.45
5	406.40	90.00	4	33.82
6	254.00	32.00	5	30.54
7	254.00	32.00	6	30.44
8	25.40	2.00		

3.2. Instancias de Prueba del Problema RDA.

Además de la instancia de prueba Alperovits, mostrada anteriormente en este trabajo, se utilizan también dos redes de prueba más. Éstas corresponden a la red de Hanoi y a la red de Balerma. La red de Hanoi es la red de tuberías para el suministro de agua de la ciudad de Hanoi (Vietnam), propuesta por Fujiwara y Khang [Fujiwara, 1987]. Esta red, de tamaño intermedio, consta de 3 lazos, 32 nodos, y 34 tuberías, Figura 3.7. Dicha red, no contiene estaciones de bombeo ya que hay una sola fuente de abastecimiento fija con una elevación de 100 metros.

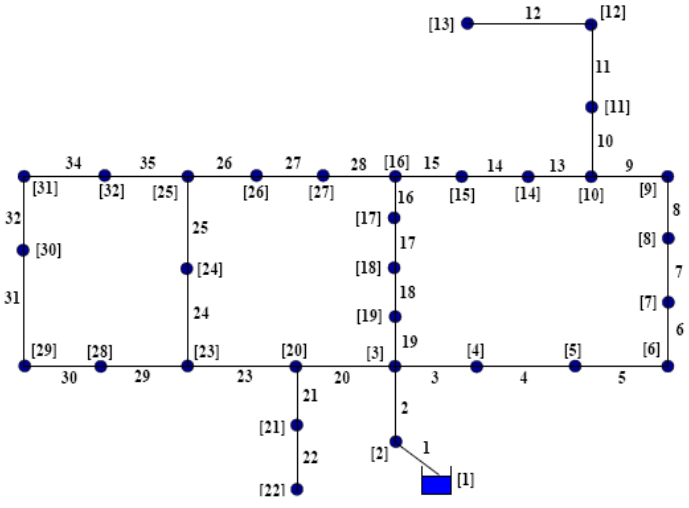


Figura 3.7 Red de Hanoi (tomada de [Baños, 2006])

El modelo que resuelve el problema de Diseño de Redes de Distribución de Agua debe cumplir una serie de restricciones tanto de diseño como de operación de la red:

1. Restricciones de diseño: se pueden escoger entre 6 diámetros comerciales, tabla 2 Apéndice I. El total de configuraciones posibles para la red Hanoi es 6^{34} = 2.865 110^{26}.
2. Restricciones hidráulicas: para que la red tenga un funcionamiento adecuado, se deben cumplir las restricciones de conservación de la masa y la energía. Así mismo, para los usuarios de la red Hanoi, la presión mínima requerida que debe cumplirse es de 30 mca.

Si alguna de las restricciones no se cumple, se tendrá una red con funcionamiento inadecuado que difícilmente podrá ofrecer a los usuarios un servicio acorde a sus necesidades. La pauta para el diseño de la red es con base en un vector de diámetros de tuberías disponibles que pueden ser asignados para cada una de las tuberías de la red. Es importante resaltar que cada elemento del vector que corresponde a un diámetro de tubería está relacionado directamente con un elemento del vector de costos comerciales. Pero, para poder resolver el problema de Diseño de Redes de Distribución de Agua es necesario también conocer las demandas de cada uno de los usuarios de la red. Para ello se tiene un vector de requisitos en el que cada elemento representa las necesidades hídricas que deben cumplirse para cada usuario.

La solución del problema de Diseño de Redes de Distribución de Agua, de la instancia Hanoi y en general de cualquier instancia de tamaño medio o grande, se obtiene en dos etapas, en este trabajo de tesis. En la primera etapa se debe encontrar una solución que cumpla con todas las restricciones de operación de la red para garantizar una operación y servicio a los usuarios adecuados, y posteriormente en una segunda etapa, se debe trabajar en obtener una solución con el menor costo para el diseño de la red. En la primera etapa se utiliza el evaluador hidráulico Epanet y en la segunda etapa es donde precisamente se hace uso de un algoritmo evolutivo, desarrollado en este trabajo de tesis, para trabajar en la optimización del problema. Cabe mencionar que la solución de problemas reales es de mayor complejidad e interés que la solución de instancias teóricas y para ello esta tesis se enfoca en la solución de una instancia de prueba real llamada Balerma, Figura 3.8.

La red de Balerma corresponde a una red de riego real de gran tamaño propuesta por Reca et al. [Baños, 2006]. Esta red se encuentra en el municipio de Balerma, provincia de Almería y suministra agua al distrito de riego Sol Poniente de la región Almeriense. Es una red con 4 presas, que contiene 443 nodos (bocas de riego), cuya agua proviene de 4 reservas. Tiene un total de 454 tuberías organizadas en 8 lazos. Para esta red se pueden elegir cualquiera de los 10 diámetros comerciales disponibles, Tabla 3 Apéndice I, por lo tanto el total de configuraciones posibles es 10^{454}. Esta red es un sistema importante puesto que se encarga de abastecer los recursos hídricos que requieren los cultivos de la ciudad de Almería. Sin embargo, es extremadamente compleja de optimizar [Reca, 2006]. La descripción de la red Balerma y la base de datos de diámetros de tuberías comerciales disponibles pueden ser descargadas desde el apartado de materiales auxiliares de la Unión Geofísica Americana (AGU) [AGU, 2000].

Figura 3.8 Red de Balerma (tomada de [Reca, 2006])

3.3. Modelo Matemático

Los experimentos realizados en este trabajo de investigación se basan en instancias de prueba teóricas que han sido utilizadas en la literatura por múltiples investigadores, véase 2.1. Así mismo, los experimentos realizados en este trabajo, utilizan la instancia de prueba real, conocida mediante el nombre de red Balerma, la cual ha sido presentada en un trabajo específico [Reca, 2006]. Las instancias, previamente descritas en el punto 3.2, resueltas por diferentes investigadores [Savic, 1997; Sherali, 1998; Montesinos, 1999; Reca, 2008] se apegan a formulaciones matemáticas en las que la función objetivo consiste en minimizar el costo del diseño de la red, partiendo del supuesto de depender solo de la inversión inicial de las tuberías. En estos trabajos, los investigadores modelan el problema de diseño de redes de distribución de agua mediante un modelo matemático que contiene las restricciones de diseño de la red juntas con las restricciones de operación de la red. En otros casos presentan las restricciones de diseño y las restricciones de operación de la red las manejan de forma implícita. Cabe mencionar que el objetivo de los trabajos es dimensionar las tuberías de la red con diámetros comerciales, mientras que la topología de la red, la conectividad y las demandas de los usuarios se encuentran previamente definidas.

En este trabajo se ha propuesto una formulación matemática que consiste de dos modelos matemáticos independientes que deben resolverse. Debido a que el problema de Diseño de Redes de Distribución de Agua consta de restricciones de diseño y de restricciones de operación, se han definido dos modelos independientes para clasificar las restricciones de diseño separadas de las restricciones de operación: el modelo de programación lineal y el modelo de satisfacción de restricciones. Así, los dos modelos propuestos permiten resolver un mismo problema garantizando el diseño de la red de menor costo y la operación adecuada de la red para brindar a los usuarios un servicio acorde a sus necesidades. En el modelo matemático de satisfacción de restricciones se tienen las restricciones que ayudan a tener un funcionamiento adecuado de la red, y un servicio aceptable para los usuarios de la misma. En este modelo se encuentran las restricciones de 1) presiones mínimas y máximas, 2) restricciones de la conservación de la energía y 3) restricciones de velocidades mínimas y máximas. Sin embargo, la mayoría

de los autores se enfocan en las dos primeras restricciones, sin considerar restricciones de velocidades, con el argumento de que las restricciones de presiones y velocidades se encuentran estrechamente relacionadas.

En este trabajo se han hecho pruebas experimentales considerando las velocidades en la red. También se hicieron pruebas sin considerar las velocidades de la red, obteniendo resultados de presiones muy parecidos.

El algoritmo evolutivo en ambiente grid, resultado de este trabajo de investigación, permite resolver instancias de prueba teóricas y prácticas de diferente tamaño, considerando todas las restricciones definidas en los modelos.

3.3.1. Modelo de Programación Lineal

El modelo de programación lineal propuesto en este trabajo, Figura 3.9, contiene las restricciones de diseño de una red de distribución de agua. La función objetivo del modelo consiste en minimizar el costo de diseño de la red, denotado mediante $C(p)$, para referirse al costo del diseño de una planeación de una red *p*.

$$Min\ C(p) = \sum_{i=1}^{n_n} \sum_{j=1}^{n_n} \sum_{d_k=d_1}^{d_n} C_{ijd_k} l_{ijd_k} X_{ijd_k} \quad (1)$$

Sujeto a:

$$\sum_{i=1}^{n_n} \sum_{j=1}^{n_n} X_{ijd_k} \geq 0 \qquad \forall (d_k = d_1,...,d_n) \quad (2)$$

$$\sum_{i=1}^{n_n} \sum_{d_k=d_1}^{d_n} X_{id_k} \geq 1 \qquad \forall (j = 1,...,n) \quad (3)$$

$$\sum_{i=1}^{n_n} \sum_{j=1}^{n_n} \sum_{d_k=d_1}^{d_n} X_{ijd_k} l_{ijd_k} = 1 \quad (4)$$

$$X_{ijd_k} \in \{0,1\} \qquad \forall (i,j,k) \quad (5)$$

Figura 3.9 Modelo de Programación Lineal para el problema RDA

Sea $T = \{t_1, t_2, t_3, ..., t_n\}$ un conjunto de tuberías de la red de distribución de agua. Sea $D = \{d_1, d_2, d_3, ..., d_n\}$ un conjunto de diámetros comerciales disponibles que

pueden ser asignados a cada tubería $t_{ij} \in T$ de la red. Sea $N = \{n_1, n_2, n_3, \ldots, n_n\}$ un conjunto de nodos de la red, que permiten la conexión de tuberías. Sea $L = \{l_1, l_2, l_3, \ldots l_n\}$ un conjunto de longitudes para asignar a las tuberías de la red.

Cada una de las tuberías de la red $t_{ij} \in T$ está conectada desde un nodo origen $n_i \in N$ hasta un nodo destino $n_j \in N$, donde *i,j* son pares ordenados que definen un segmento de tubería $t_{ij} \in T$. Cada una de esas tuberías $t_{ij} \in T$ de la red, tiene una longitud $l_{ij} \in L$ y un diámetro $d_k \in D$. De forma implícita existe una restricción que se refiere a la longitud de las tuberías y al tamaño de los diámetros que pueden asignarse a las tuberías: $\mathrm{L_{ij}} \succ 0$ y $\mathrm{d_k} \in \mathrm{D} \succ 0$, respectivamente. Para cada instancia de prueba a resolver el tamaño de los diámetros de las tuberías de la red y las longitudes de las tuberías de la red se encuentran previamente definidas. Es importante notar que únicamente un diámetro de los $\mathrm{d_k} \in \mathrm{D}$ puede asignarse a cada una de las tuberías $t_{ij} \in T$. Otro punto importante es que tres o más nodos conectados forman una malla o ciclo. Todas las mallas que existen en una red forman el conjunto M.

La ecuación (1) es la función objetivo y consiste en obtener el costo mínimo de la red en base al costo $\mathrm{cd_k}$ de cada tubería y a la longitud $l_{ij} \in L$ de cada segmento de la red. La definición de la función de costo se hace con base en los costos de cada diámetro que se asigna a una tubería y también con base en la longitud de dicha tubería. El conjunto de restricciones en (2), del modelo matemático, indican que una o más tuberías de la red pueden tener el mismo diámetro $d_k \in D$. Esto es porque en la configuración de la red se pueden asignar diámetros repetidos para diferentes tuberías. Incluso puede ocurrir que toda la configuración de la red este formada por un solo diámetro de tubería, aunque esto en la práctica es muy poco común. Al mismo tiempo el conjunto de restricciones en (2) indica cuando un diámetro, incluido en el conjunto *D* de diámetros comerciales es utilizado en una tubería. El conjunto de restricciones en (3) indican que a cada nodo de la red, $n_i \in N$, puede conectarse una tubería de longitud $l_{ij} \in L$ con el mismo o diferente tamaño de diámetro de tubería tomado de la lista de diámetros comerciales $d_k \in D$. El conjunto de restricciones en (4) indican que para cada tubería de longitud $l_{ij} \in L$, debe

utilizarse exclusivamente un diámetro $d_k \in D$. Finalmente, el conjunto de restricciones en (5) define valores que pueden asignarse al conjunto de variables *X*. Por ejemplo, tomando como referencia la ecuación (1), si una tubería conectada $t_{ij} \in T$ utiliza un diámetro $d_k \in D$ entonces X_{ijdk} = 1 de otra forma X_{ijdk} = 0.

Una solución válida, para el problema de Diseño de Redes de Distribución de Agua, se obtiene cuando para cada una de las tuberías de la red se asigna exactamente uno de los diámetros $d_k \in D$. La asignación de los diámetros de tuberías a las tuberías de la red se refiere a una planeación del diseño de la red $\mathrm{p(d_k) \in T_i}$ a la cual, en lo sucesivo, se le denominará configuración de la red. De esta forma, el objetivo del problema de Diseño de Redes de Distribución de Agua es encontrar una configuración *p* que minimice la función de costo.

3.3.2. Modelo de Satisfacción de Restricciones

El modelo de satisfacción de restricciones Figura 3.10, que se compone por las restricciones de operación de la red, contiene las restricciones hidráulicas que deben cumplirse para que la red opere correctamente y los usuarios tengan un servicio adecuado a sus necesidades.

$$\sum_{n_i} Q_{in} - \sum_{n_i} Q_{out} = Q_e \qquad \forall\ n_i \in N \qquad (6)$$

$$\sum_{m} h_L = \sum_{m} E_p \qquad \forall m \in M \qquad (7)$$

$$H_{\min} \le Hn_i \le H_{\max} \qquad \forall\ n_i \in N \qquad (8)$$

$$V_{\min} \le Vt_{ij} \le V_{\max} \qquad \forall\ t_{ij} \in T \qquad (9)$$

Figura 3.10 Modelo de Satisfacción de Restricciones

El conjunto de restricciones en (6) se refiere a la ley de la conservación de la masa. Esta expresión indica que los caudales que entran en un $n_i \in N$ y los caudales que salen de dicho nodo deben estar equilibrados con las demandas o aportes de la red. La sumatoria de flujos que entran y salen de un nodo es igual a cero.

El conjunto de restricciones en (7) se refiere a la ley de la conservación de la energía en una malla *M* o entre dos nodos de cabecera conocida que definen un ciclo abierto. Indica que la sumatoria de las pérdidas de energía por fricción a lo largo de los tramos que pertenecen a una malla debe ser igual a cero, en caso de que no existan bombas de potencia en la malla.

El conjunto de restricciones en (8) hace referencia a las presiones permitidas en el diseño de la red de distribución de agua. La presión en un nodo $n_i \in N$ debe ser mayor que la presión mínima requerida y menor que la presión máxima requerida, mismas que se definen de acuerdo con las características del problema. Por ejemplo para la red Alperovits las normas de presiones mínimas son valores de 30 mca, mientras que los valores de las presiones máximas son 100 mca. En cuanto a los usuarios, las presiones mínimas garantizan un servicio adecuado evitando desperdicios con presiones mayores a las definidas y un servicio inadecuado con presiones inferiores a las definidas en el intervalo. En cuanto a la red, las presiones mínimas y máximas aseguran el funcionamiento correcto de la misma evitando rupturas de tuberías con presiones elevadas.

El conjunto de restricciones en (9) se refiere a las velocidades permitidas en la red. La velocidad del flujo en una tubería $p_{ij} \in T$ debe ser mayor que la velocidad mínima requerida y menor que la velocidad máxima requerida, generalmente definidas de forma estándar en un intervalo de 0.05 a 5 m/s. Las velocidades definidas, permiten el funcionamiento adecuado de la red evitando sedimentos y con ello la reducción de diámetros de tuberías cuando son menores a las velocidades mínimas. Las velocidades máximas evitan que en la red haya pérdidas de carga que afectan las presiones de la red de distribución de agua.

Es importante mencionar que para tener soluciones válidas para el problema de Diseño de Redes de Distribución de Agua es necesario que se resuelvan ambos modelos matemáticos. En caso de resolver únicamente el modelo de programación lineal las

soluciones son de bajo costo pero generalmente no cumplen los requisitos definidos en el modelo de satisfacción de restricciones. Así mismo, si sólo se resuelve el modelo de satisfacción de restricciones entonces se obtienen soluciones que satisfacen la operación de la red y las necesidades hídricas de los usuarios, pero el diseño de la red no será de costo mínimo. Ambos objetivos deben cumplirse considerando que éstos se contraponen, aunque algunos autores consideran que las restricciones de presiones del modelo hidráulico están estrechamente relacionadas con las restricciones de velocidades [Reca, 2006], por lo que se limitan a verificar que las soluciones cumplan las restricciones de presiones mínimas del modelo.

Capítulo 4. Metodología de Solución

En este capítulo, se hace una descripción de las arquitecturas y herramientas de cómputo paralelo disponibles que se utilizan para el diseño y desarrollo del algoritmo mostrado en el capítulo 5, mencionando de cada una de ellas sus ventajas y desventajas. Se hace también una descripción detallada de las técnicas heurísticas existentes, haciendo énfasis en algoritmos evolutivos por ser concretamente la técnica implementada en este trabajo de tesis. Se describen también, estudios experimentales acerca del espacio de búsqueda, de la población inicial, de los operadores y de los valores paramétricos definidos en el algoritmo evolutivo secuencial. Finalmente, se describe el modelo de paralelización definido en este trabajo y utilizado en el algoritmo evolutivo en ambiente Grid desarrollado en esta tesis doctoral.

4.1. Algoritmo Evolutivo

El Algoritmo Evolutivo llamado PEA-WDND desarrollado en este trabajo, mediante la metodología de la Figura 4.1, es una técnica probabilística que converge a una solución factible, que incluso puede ser la óptima, mediante una serie de iteraciones (I), llamadas generaciones. El número de generaciones (NG) es definido mediante un valor específico que indica el criterio de paro del Algoritmo Evolutivo. El criterio de paro indica el momento en que el Algoritmo Evolutivo debe finalizar.

En cada una de las generaciones del Algoritmo Evolutivo, los individuos son evaluados para obtener su aptitud basándose en la función objetivo del problema de Diseño de Redes de Distribución de Agua.

El algoritmo PEA-WDND trabaja con una población de individuos, en la cual cada individuo representa una solución potencial para el problema a resolver. El algoritmo evolutivo desarrollado:

1) Utiliza reglas de transición probabilísticas en lugar de reglas de transición deterministas.
2) Utiliza poblaciones de individuos que lo hace menos sensible de quedar atrapado en óptimos locales.

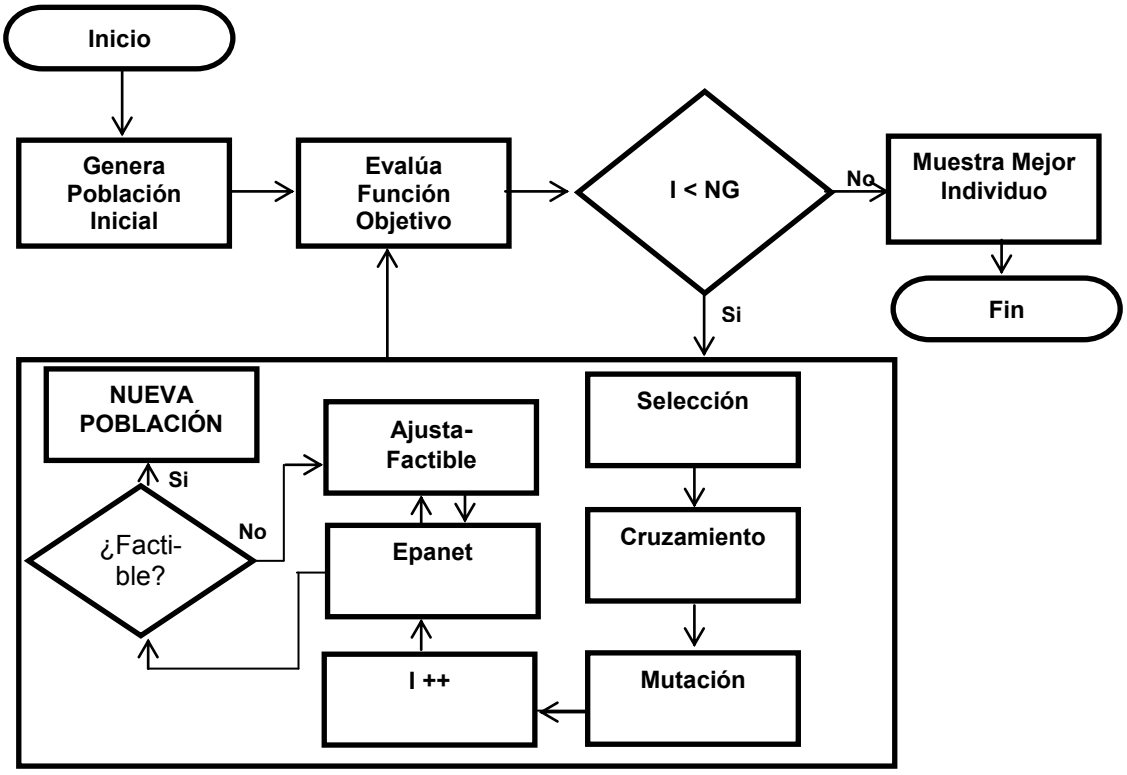


Figura 4.1 Diagrama de flujo del Algoritmo Evolutivo SEA-WDND

El primer paso para resolver el problema de Diseño de Redes de Distribución de Agua, mediante el Algoritmo Evolutivo PEA-WDND, es determinar la codificación de los individuos. Tradicionalmente, en otros algoritmos la codificación de los individuos se hacía mediante una representación binaria de las variables de decisión, convirtiendo cada variable que interviene en el problema en su equivalente valor binario. Sin embargo, la representación binaria representa sobrecarga del algoritmo con operaciones de codificación y decodificación de los valores del cromosoma. Así mismo se sabe que esta codificación presenta el problema de pérdida de precisión en la información que se utiliza.

La codificación de enteros o reales consiste en yuxtaponer cada una de las variables del problema, dando lugar a una cadena de variables para representar el problema. De acuerdo con [Reca, 2006] esta representación presenta más ventajas que la representación binaria, la principal es eliminar las operaciones de codificación y decodificación y con ello mejorar la velocidad de ejecución del algoritmo, así que la representación de valores reales es la representación que se utiliza para la codificación de los individuos en este trabajo.

Una vez que se define la representación de la solución se procede a la creación de la población inicial, la cual puede hacerse de diferente forma, dependiendo de la naturaleza del problema e incluso del tamaño de la instancia. La población inicial del Algoritmo Evolutivo PEA-WDND puede crearse de forma aleatoria o de forma determinista, según se requiera (en instancias pequeñas como Alperovits la creación de la población aleatoria es adecuada, en instancias como Balerma es más adecuada la creación determinista de la población inicial.

La población inicial creada de forma aleatoria puede contener individuos factibles e infactibles y mediante la evolución de la población se pueden seleccionar los mejores individuos para ofrecer una solución al problema. Sin embargo, en ocasiones no es trivial acercarse a soluciones factibles partiendo de poblaciones infactibles (resultados de los estudios realizados en este trabajo, capítulo 6) y para evitar el problema se recurre a la creación de la población de forma determinista, la cual consiste en la asignación específica de los genes del individuo con la finalidad de crear individuos con características deseables, mejor conocidos como individuos factibles. Dichos individuos pueden tener aptitudes cercanas a la solución óptima, pero para esto se debe conocer con anticipación la naturaleza del problema. En este trabajo se hicieron estudios experimentales para realizar la creación de la población inicial de forma aleatoria y de forma determinista, siempre considerando que existiera diversidad en los individuos, véase 5.2.2. Una vez creada la población, el Algoritmo Evolutivo PEA-WDND determina el valor de la función objetivo para conocer la aptitud de cada uno de los individuos de la población. Posteriormente, se ordena la población, de acuerdo con las aptitudes de los individuos, mediante el método QuickSort [Hoare, 1962], cuya complejidad computacional es de $O(n)^2$. De la población ordenada, se extrae una muestra que contiene a los

individuos con mejores aptitudes, definidas mediante la función de aptitud. La función de aptitud, en problemas de minimizar, se relaciona con la inversa de la función objetivo del problema y se utiliza como métrica para evaluar a los individuos. En el caso del problema abordado, la función objetivo determina que los mejores individuos son aquellos que presenten los costos más bajos para la configuración de la red de distribución de agua. Mientras que la función de evaluación determina que los individuos con función objetivo baja tienen una función de evaluación alta, ya que los individuos con costos más bajos son individuos con mayor aptitud.

En el Algoritmo Evolutivo AE-WDND la probabilidad de selección de un individuo se basa en la aptitud relativa que presenta respecto a otros individuos. Así, con base en los valores de aptitud se determina cuáles son los individuos que sobreviven en una generación y continúan existiendo a través de sus descendientes mediante el operador de selección. El diagrama 4.1 muestra que el objetivo del Algoritmo Evolutivo es encontrar el mejor individuo de una población, con base en la aptitud de cada uno de ellos. Este individuo es la mejor solución para el problema abordado y se elige después de un proceso de evolución de la población en el que intervienen los operadores de selección cruce y mutación, sin perder de vista que los individuos deben cumplir también las restricciones del modelo hidráulico definidas en el modelo de satisfacción de restricciones, véase 3.3.2. Finalmente, se aplican de manera iterativa los operadores de selección, cruce y mutación hasta alcanzar el criterio de optimización o de paro del algoritmo, definido en el presente trabajo mediante el número de generaciones.

4.1.1. Métodos de Selección

El operador de selección tiene una función importante dentro del algoritmo evolutivo PEA-WDND, ya que se encarga de elegir a los individuos que se convertirán en padres para continuar así la evolución de la población. Este operador permite aumentar la presencia de los individuos mejor adaptados a lo largo de las generaciones. No obstante, con la selección de los individuos mejor dotados se observa la pérdida de la diversidad de los individuos de la población. Para introducir variabilidad en la población se recurre, en este trabajo de tesis, a los operadores de cruce y de mutación.

Algunos parámetros que determinan la eficiencia y la precisión del operador de selección se conocen como sesgo, amplitud y complejidad computacional, entre otros. El sesgo es el valor absoluto de la diferencia de la probabilidad real que tiene un individuo de ser seleccionado y la probabilidad esperada. La amplitud se refiere al número de veces que un individuo puede seleccionarse para reproducirse. La complejidad computacional se establece con base en el tiempo que se emplea para el muestreo de la población. Como en todo algoritmo, es deseable que la complejidad computacional del operador de selección sea una función lineal representada por O(n), donde *n* es el tamaño de la población.

El operador de selección, para el algoritmo PEA-WDND, se implementa utilizando algunas de las técnicas definidas en la literatura. Entre las técnicas de selección más comunes se encuentran: ***la selección por ruleta*, *la selección por torneo* y *la selección aleatoria*** [Amor, 2008]. Cada uno de estos métodos de selección tiene sus propias características, ventajas y desventajas que a continuación se describen.

4.1.1.1. Método de Selección Ruleta.

La selección por ruleta, propuesta por [De Jong, 1990], pertenece a un grupo de técnicas conocidas como selección proporcional [Holland, 1975].

El método de selección por ruleta consiste en elegir a los individuos basándose en su valor de aptitud. Este método representa la aptitud de cada individuo como un porcentaje de una sección de la ruleta. La probabilidad, $p(c_i)$ de que un individuo c_i, sea seleccionado está dada por la fórmula: $p(c_i) = \frac{f(c_i)}{\sum_{j=0}^{n} f(c_j)}$ donde el numerador es la aptitud de un individuo obtenida mediante una función de aptitud, y el denominador se refiere a las probabilidades acumuladas de cada individuo obtenidas con la suma de las probabilidades del individuo de interés y de sus antecesores una vez que se ordenan, en forma decreciente. En el método de selección por ruleta se calcula una aptitud media de los individuos de la población y para cada individuo se asigna una probabilidad esperada de selección, que sirve para saber qué probabilidad existe de que un individuo sea seleccionado. En general, el método de ruleta consiste en lo siguiente: los individuos son colocados de forma contigua formando segmentos angulares de un círculo, Figura 4.2.

El tamaño de cada segmento es proporcional a la aptitud relativa de cada individuo con respecto a la aptitud del resto de la población. En este método, entre mayor sea la adaptación del individuo, mayor es el sector que le corresponde en la ruleta (candidato 1) y consecuentemente, mayor es la probabilidad de que éste sea elegido. Una vez definida la aptitud de los individuos, se gira la ruleta para elegir al individuo que se encuentre en el segmento seleccionado cuando la ruleta se detenga. El proceso se repite hasta obtener el número de individuos que van a reproducirse.

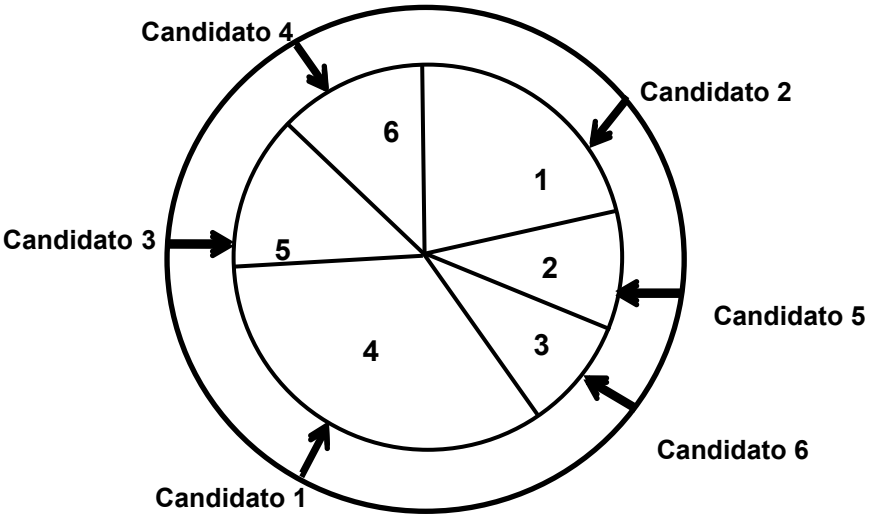


Figura 4.2 Método de la Ruleta

Por ejemplo, se tienen 6 individuos (I_1, I_2, I_3, I_4, I_5, I_6). La aptitud de los individuos es, I_1=600, I_2=300, I_3=290, I_4=819, I_5=542, I_6=496. La suma de las aptitudes de la población es de 3047. La probabilidad de selección resultante de dividir cada aptitud entre la suma de las aptitudes puede verse en la tabla 4.1.

Tabla 4.1 Probabilidades de selección con método ruleta

Individuo	Aptitud	Probabilidad de Selección	Probabilidad Acumulada
I_4	819	0.269	0.269
I_1	600	0.197	0.466
I_5	542	0.178	0.644
I_6	496	0.163	0.806
I_2	300	0.098	0.905
I_3	290	0.095	1.000

Las ventajas de esta técnica de selección son las siguientes. La selección por ruleta es un método simple con complejidad computacional de $O(n)^2$; es fácil de implementar y maneja un sesgo optimo, que significa que la probabilidad real que tiene un individuo para reproducirse es igual que la probabilidad esperada. Sin embargo, este método presenta también algunas desventajas importantes: el método de selección por ruleta no garantiza una mínima amplitud, por lo que los individuos más aptos pueden seleccionarse más de una vez para reproducirse ocasionando que haya múltiples réplicas de un individuo en las siguientes generaciones. Incluso, al ser una técnica de selección estocástica, puede ocurrir que un individuo con aptitud alta no sea seleccionado para reproducirse. Además presenta el problema de que los individuos con menor aptitud rara vez son seleccionados para reproducirse y esto reduce paulatinamente la diversidad de la población y la posibilidad de explorar algunos espacios de búsqueda. Adicionalmente, este método de selección resulta ineficiente a medida que incrementa el tamaño de la población ya que la población debe ser ordenada para poder hacer la selección con base en su aptitud.

4.1.1.2. Método de Selección por Torneo.

El método de selección por torneo consiste en elegir un subconjunto de *n i*ndividuos de una población, donde *n* puede tomar valores de dos o más individuos [Baños, 2006. Es un método eficiente y fácil de implementar, con complejidad computacional del algoritmo de O(n). Esta técnica presenta como ventaja respecto a otras técnicas de selección su eficiencia computacional ya que no requiere escalamiento de la función de aptitud (usa comparaciones directas) por lo cual no requiere la ordenación de los individuos para poder seleccionarlos con base en su aptitud. Además, este método puede implementarse en paralelo.

El método de selección por torneo consiste en lo siguiente: los individuos de una población compiten con su aptitud para ser seleccionados. Cuando el valor de *n* crece, los individuos tienen mayor dificultad de ser seleccionados puesto que se elige como ganador al individuo con mayor aptitud. El método trabaja mediante comparaciones directas de los individuos de una población para seleccionar, del subconjunto, al individuo con mayor aptitud [Amor, 2008]. El proceso se repite hasta obtener un número suficiente de individuos que van a reproducirse, para formar una nueva población. Típicamente, el número de individuos que participan en el torneo es bajo (2-5). Si el número de individuos

que participan en el torneo incrementa se pierde en gran porcentaje la diversidad genética ocasionando convergencia prematura del algoritmo [Ghanea, 2003]. Por el contrario, si el número de individuos que participan en el torneo es bajo siempre la optimización puede tornarse excesivamente lenta [Horn, 1994]. El objetivo entonces, es encontrar un balance entre el tiempo de optimización y la convergencia prematura del algoritmo.

El método de selección por torneo puede ser de tipo determinista o probabilista. En el método determinista el mejor individuo puede ser seleccionado *p* veces; en el método probabilista en lugar de elegir siempre al individuo de mejor aptitud, se aplica el operador prob $(p)^2$ y si el resultado es cierto se selecciona al más apto. De lo contrario se selecciona al individuo menos apto. El valor de p permanece fijo a lo largo de todo el proceso evolutivo ($0.5 < p \leq 1$).

4.1.1.3. Método de Selección Aleatoria.

El método de selección aleatoria es un método estocástico que se consiste en elegir al azar a un individuo de una población. Este tipo de selección permite que cualquier individuo de una población pueda ser elegido, lo cual puede considerarse como ventaja o desventaja dependiendo de la naturaleza del problema y del objetivo que se persiga al generar nuevas poblaciones. Se dice que la selección aleatoria es el método de muestreo probabilístico a partir del cual han evolucionado otros métodos de selección más complejos. La tabla 4.2 muestra una comparación de los diferentes métodos de selección que pueden implementarse. Se muestran las ventajas y desventajas de cada método y se describe en qué casos se recomienda su uso.

Tabla 4.2 Métodos de Selección de Individuos

Método	Descripción	Ventajas	Desventajas
Ruleta	Selecciona a los individuos basándose en su valor de aptitud. Se sitúan a los individuos de forma contigua formando segmentos angulares de un círculo. La complejidad del algoritmo es $O(n)^2$	Es un método simple que puede implementarse con relativa facilidad.	El individuo más apto puede ser seleccionado varias veces ocasionando una convergencia prematura. No es recomendable para poblaciones de gran tamaño.
Elitista	Selecciona a los mejores cromosomas y los copia directamente en la nueva población.	Evita que se pierda la mejor solución.	La convergencia puede ser prematura en el caso de funciones multimodales.
Torneo	Determinista. Permite seleccionar en base a comparaciones directas de los individuos. Probabilístico. Es idéntico al torneo determinista, excepto que en lugar de elegir siempre al individuo de mejor aptitud, se aplica el operador $prob\ (p)^2$ y si el resultado es cierto se selecciona al más apto. De lo contrario se selecciona al menos apto. La complejidad del algoritmo es O(n)	Puede implementarse en paralelo. Es una técnica eficiente y fácil de implementar. Usa comparaciones directas.	En la versión determinista puede ocurrir que a los individuos menos aptos no se les da la oportunidad de sobrevivir.
Aleatoria	Asigna la misma probabilidad de selección a cada uno de los individuos de la población.	Todos los individuos tienen probabilidad de ser seleccionados.	Todos los individuos tienen probabilidades de sobrevivir y reproducirse, incluso aquellos cuya aptitud es baja.

4.1.2. Métodos de Cruce

En Algoritmos Evolutivos, el operador de cruce consiste en la combinación de soluciones para generar nuevos individuos. El operador de cruce es un operador importante en el algoritmo PEA-WDND, ya que consiste en combinar información genética procedente de dos individuos durante el proceso de reproducción. El operador de cruce utiliza dos cromosomas y combina la información genética con la finalidad de generar descendientes que tengan mejor aptitud que los progenitores, de tal forma que los individuos mejor adaptados pueden continuar existiendo por generaciones a través de los descendientes.

El operador de cruce es el operador principal de los algoritmos evolutivos. Es una parte esencial del algoritmo evolutivo PEA-WDND porque, con apoyo del operador de selección, permite la exploración del espacio de búsqueda. Es decir, el operador de cruce permite trasladar la búsqueda de soluciones hacia zonas alejadas del espacio de búsqueda. Este operador, en gran parte, es el responsable de la evolución de la población y produce la convergencia del algoritmo PEA-WDND. El estudio del operador de cruce, en el presente trabajo ha sido abordado desde una doble perspectiva. Por un lado la idea de combinar dos individuos para generar dos descendientes y por otro lado el mecanismo que se utiliza para implementar la combinación de individuos.

En los inicios de los algoritmos evolutivos, la forma de combinación de los individuos se realizaba sobre cadenas binarias utilizando el mecanismo más simple de cruce: ***el cruce monopunto*** [Holland, 1975]. Sin embargo, al igual que con los demás operadores del algoritmo evolutivo también han surgido otras técnicas de combinación de individuos que buscan adaptarse al tipo de codificación obteniendo una mayor eficiencia. Algunos tipos de cruce, adicionales al cruce monopunto, son ***el cruce multi-punto*** [Goldberg, 1989] y ***el cruce uniforme*** [Syswerda, 1989]. En el algoritmo PEA-WDND se ha implementado el cruce monopunto y multipunto, mismos que a continuación se describen.

4.1.2.1. Método de Cruce Mono-punto

Esta técnica consiste en elegir de manera aleatoria un escalar k ∈ [1, l) que corresponde a un gen del cromosoma con longitud l. El escalar k se conoce como punto de cruce y es precisamente el punto a partir del cual dos cromosomas, de los individuos, se dividen para intercambiar genes y generar descendientes, Figura 4.3. Como puede observarse en la Figura 4.3, el punto de cruce es en el gen 4 del cromosoma. De esta forma, el Descendiente 1 está formado por los *k* primeros genes del Individuo 1, y de *k* a *l* genes del individuo 2. El descendiente 2 se forma por *k* primeros genes del Individuo 2, y de *k* a *l* genes del individuo 1. El punto de cruce generado de forma aleatoria define que, en este caso, los cromosomas se dividan a la mitad.

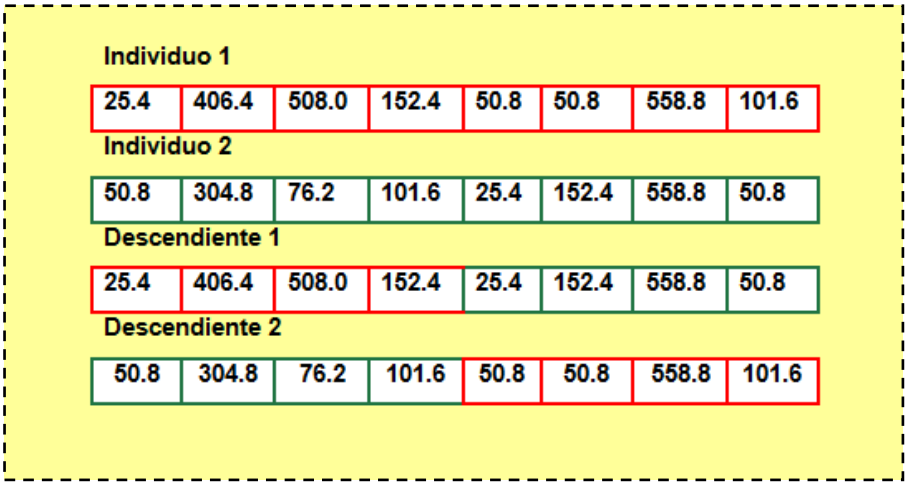


Figura 4.3 Cruce Mono-punto

La ventaja de esta técnica es su sencillez de funcionamiento e implementación. En cromosomas de longitud pequeña (para representar red Alperovits o Hanoi) este esquema permite obtener buenos resultados, véase capítulo 5. Sin embargo, para cromosomas con longitud grande, como el caso de la red Balerma, esta técnica no permite que las características importantes de los padres se hereden a los hijos.

4.1.2.2. Método de Cruce Multi-punto

El cruce multipunto, es el operador de cruce más común en los algoritmos evolutivos. Es una técnica que consiste en escoger de forma aleatoria ***n*** puntos de cruce, para obtener la descendencia por combinación alternada de los genes de los progenitores. El valor que *n* puede tener es mayor que dos, ya que si *n* tuviera el valor de 1 se estaría haciendo referencia al cruce monopunto, aunque en la literatura algunos autores expresan que la forma más simple del cruce multipunto es el cruce de un punto [Amor, 2008].

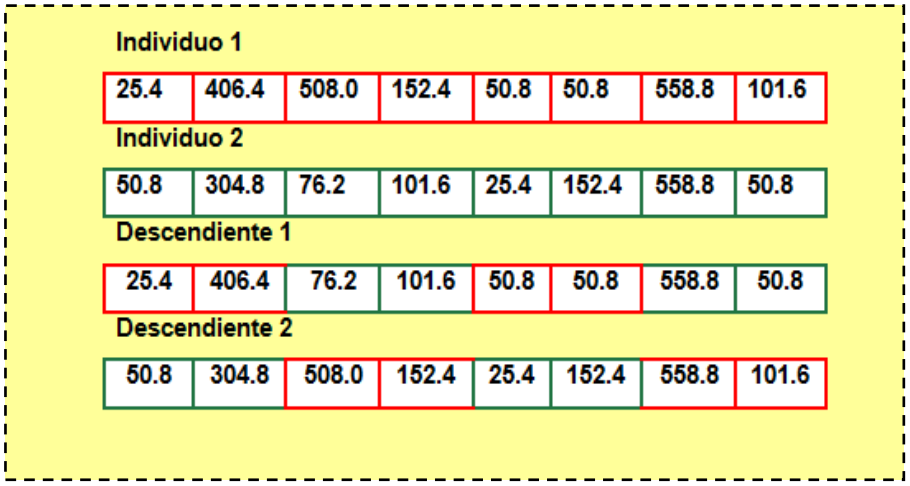


Figura 4.4 Cruce Multipunto

En la Figura 4.4 se muestran dos individuos que intercambian porciones contiguas del cromosoma para formar nuevos descendientes. El punto de cruce se genera de forma aleatoria. En realidad, el punto de cruce ***n*** de la figura 4.4 está definido por tres números aleatorios que corresponden a los valores 2,4 y 6. Aumentar el valor de *n* supone realizar una búsqueda más exploratoria que explotadora. Sin embargo, al aumentar el valor de *n* se incrementa también el tiempo de cálculo necesario para generar la descendencia.

[Goldberg, 1989] realiza una serie de estudios experimentales utilizando el operador de cruce multipunto y concluye que si se introduce un número excesivo de puntos de corte, se reduce la eficacia del algoritmo.

4.1.2.3. Método de Cruce Uniforme

El cruce uniforme es una técnica similar al cruce multipunto que consiste en escoger de forma aleatoria ***n*** puntos (n≥1 y n≤ Número de genes del individuo) de cruce de un cromosoma, para obtener la descendencia, Figura 4.5.

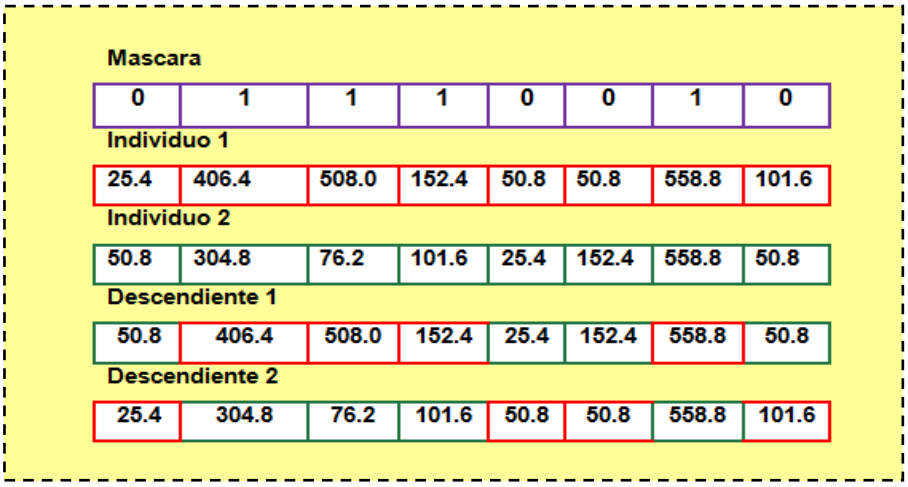


Figura 4.5 Cruce Uniforme

A diferencia del cruce multipunto, en donde se intercambian porciones contiguas del cromosoma, en este tipo de cruce se intercambian alelos separados [Amor, 2008]. La generación de la descendencia es guiada por una máscara de cruce, la cual consiste de un vector binario, con la misma longitud que el cromosoma a cruzar, generado aleatoriamente. En el vector binario se define la probabilidad de elegir un gen del cromosoma 1, o del cromosoma 2 para la creación de nuevos descendientes. Es decir, el descendiente 1 adopta el valor de los genes del individuo 1 en las posiciones en las que la máscara contenga un valor '1' y en las posiciones de la máscara de cruce que tengan un valor '0', el descendiente obtiene los genes del individuo 2. El segundo descendiente adopta en cada posición los valores del progenitor contrario al del primer descendiente. Así cada uno de los descendientes está formado por una cantidad aleatoria de genes de los individuos progenitores.

De acuerdo con los estudios presentados en la literatura, no se encuentra una demostración que indique cual punto de cruce ofrece mejores resultados, ya que cada uno de ellos presenta ventajas y desventajas, Tabla 4.3. La elección del tipo de cruce es

más bien una decisión que dependerá de la naturaleza del problema. Se dice también que la técnica de cruce elegida no afectará la eficacia, pero si eficiencia del algoritmo, por eso es realmente importante identificar principalmente la estrategia a utilizar basándose principalmente en tamaño de la población y en el tamaño de los cromosomas. En este trabajo de tesis los métodos de cruce utilizados son monopunto y multipunto.

Tabla 4.3 Métodos de Cruce

Método	Descripción	Ventajas	Desventajas
Monopunto	Consiste en dividir un cromosoma en dos segmentos, para combinarlos con los segmentos de otro cromosoma para generar dos descendientes.	Es un método sencillo de concebir e implementar.	Recomendable para cromosomas de longitud limitada.
Multipunto	Consiste en dividir un cromosoma en tres o más segmentos para intercambiarlos con segmentos de otro cromosoma, generando normalmente dos descendientes.	Al incrementar el número de puntos de cruce se consigue una exploración más profunda del espacio de búsqueda. El cruce de dos puntos es adecuado para poblaciones grandes.	Al incrementar el número de puntos de cruce se incrementa también el tiempo para generar los descendientes.
Uniforme	Consiste en intercambiar genes, basándose en la estructura de una máscara de cruce que indica cuales genes intercambiar.	Permite preservar el esquema de los padres porque contiene una mezcla de los genes de los padres.	Se debe generar una máscara nueva para cada cruce que se realice, por lo cual no es recomendable para usarse en poblaciones grandes.

4.1.3. Métodos de Mutación

El operador de mutación, también presente en el Algoritmo Evolutivo PEA-WDND, consiste en la alteración de algunos de los genes de un cromosoma. Este operador permite la explotación del espacio de búsqueda mediante búsquedas exhaustivas en zonas específicas de dicho espacio, permitiendo la variabilidad genética, entre los individuos de diferentes generaciones. Este operador permite contrarrestar el efecto de convergencia prematura ocasionada principalmente por el operador de selección. La principal función de este operador es realizar tareas de diversificación en la población habilitando la búsqueda de soluciones alternativas.

El grado en el que actúan los operadores de cruce y mutación, en el algoritmo PEA-WDND, es regulado mediante probabilidades definidas para cada uno de ellos. La probabilidad de cruce, normalmente se define en un intervalo de [0.6-0.95] y la probabilidad de mutación en un intervalo [0-1], como lo muestran diferentes trabajos presentados en el punto 2.1 de esta tesis. El ajuste de estos parámetros, mostrado en 6.2, controla el grado en el que actúan los operadores dentro del algoritmo PEA-WDND. Los valores definidos, en los parámetros, reflejan la preponderancia del operador de cruce respecto al operador de mutación.

En la literatura, existe controversia sobre la importancia relativa del uso de los operadores de cruce y mutación. El operador de mutación generalmente suele estar presente en todas las implementaciones de los algoritmos evolutivos. Incluso, los resultados empíricos muestran que para determinadas funciones objetivo aplicar el operador de mutación puede ser suficiente para obtener resultados aceptables [Hinterding, 1995]. Sin embargo, para funciones objetivo en las que se alto grado de epistasis (múltiples óptimos locales) se dice que es indispensable aplicar el operador de cruce. En este trabajo se hicieron estudios experimentales para observar el comportamiento del algoritmo con diferentes valores para la probabilidad de cruce y mutación, véase 6.2.

De la metodología utilizada, la combinación de operadores genéticos y técnicas de selección permite observar que la implementación del algoritmo evolutivo secuencial, desarrollado en este trabajo, implica una carga computacional enorme tanto en almacenamiento de los datos procesados y resultantes como en la realización de cálculos y operaciones. Para resolver este problema de tiempo y espacio se implementa el algoritmo PEA-WDND, mediante la técnica de paralelización de algoritmos, que a continuación se describe.

4.2. Paralelización de Algoritmos

La paralelización de algoritmos es una técnica que se utiliza para abordar problemas de gran tamaño y complejidad en los cuales encontrar una solución resulta una tarea costosa. Esta técnica consiste en la división de un problema completo en problemas más pequeños de tal forma que éstos puedan resolverse computacionalmente de manera más rápida por un conjunto de procesos, siendo un proceso la unidad mínima de procesamiento que se encuentra en una computadora. La característica principal del cómputo paralelo es que el programador puede tener acceso a un conjunto de núcleos, para indicarles cuantas tareas o procesos se asignarán a cada uno de ellos. En particular, en la plataforma grid que se utiliza para las pruebas experimentales del presente trabajo se tienen 66 núcleos en el clúster Cuexcomate y 66 núcleos en el clúster Texcal, que pueden utilizarse para la asignación de tareas. Es importante mencionar que la interacción entre procesos o núcleos queda definida por el modelo de paralelización descrito en el punto 5.4. La paralelización de algoritmos se ha utilizado con éxito para resolver problemas de optimización y cada vez se utiliza con mayor frecuencia por diferentes investigadores [Talbi, 2002; Talbi, 2009; Cruz, 2009c; Cruz, 2010a; Crainic, 2010; Lorente, 2012]. El objetivo principal, del cómputo paralelo, es lograr la ejecución eficiente de algoritmos que tienen altas necesidades de recursos computacionales. La esencia de la paralelización es la cooperación de procesos, misma que permite reducir el tiempo para solucionar un problema, e incluso en algunos casos permite también mejorar la calidad de las soluciones. Este hecho tiene sus fundamentos en la gran cantidad de trabajos publicados en la literatura que utilizan heurísticas paralelas para la solución de diferentes problemas [Alba, 2005]. La utilidad del cómputo paralelo puede apreciarse,

principalmente, cuando se implementan algoritmos para abordar problemas NP-Completos en los que se resuelven instancias de prueba reales de gran tamaño, como es el caso del algoritmo evolutivo PEA-WDND, desarrollado en este trabajo para la solución de la instancia Balerma.

Con la finalidad de tener un algoritmo paralelo funcionando en ambiente Grid, es necesario definir el modelo de paralelización a utilizar (en este trabajo se utiliza un modelo propio propuesto en 4.3.3), el cual implica considerar algunos aspectos importantes. Por ejemplo, es necesario definir el número de procesos que utilizarán, la distribución del trabajo entre los procesos disponibles, la forma en la que los procesos cooperan para realizar un trabajo en menor tiempo, la comunicación de los datos entre los procesos, la sincronización para coordinar la comunicación entre procesos, entre otros. El número de procesos a utilizar en este trabajo, está limitado por la arquitectura de cómputo paralelo en la que se implementa el algoritmo paralelo; la distribución del trabajo y la cooperación entre procesos están especificadas, en gran medida, a través modelo que se utiliza para el diseño del algoritmo paralelo. Finalmente, la comunicación entre procesos se define también con base en la naturaleza del algoritmo y con base en las opciones de comunicación que ofrece el entorno a utilizar. En el algoritmo PEA-WDND, las comunicaciones esporádicas favorecen el desempeño del algoritmo, véase 4.3.3.

Actualmente, existen técnicas para el diseño de algoritmos paralelos que indican cómo puede ser paralelizado un algoritmo de acuerdo con su estructura. Así mismo, indican cómo puede modificarse un algoritmo secuencial existente de tal forma que pueda ejecutarse en paralelo [Cruz, 2010c]. En general, estas técnicas definen la división de tareas entre los procesos disponibles. En la literatura, existen dos técnicas para la paralelización de algoritmos conocidas como *paralelismo de grano grueso* y *paralelismo de grano fino*.

El paralelismo de grano grueso consiste en realizar de manera paralela múltiples ejecuciones del algoritmo secuencial de tal forma que, a pesar de utilizar mayores recursos de diferentes computadoras, se pueden obtener mejores resultados en menor tiempo. Este tipo de paralelismo se utiliza cuando en una aplicación se tienen tareas definidas que pueden ejecutarse como procesos independientes en más de un

procesador. Es decir, en este tipo de paralelismo se ejecutan diferentes operaciones de manera concurrente porque los programas pueden dividirse en subprogramas relativamente independientes. Dichos subprogramas pueden ser ejecutados por diferentes procesos sin necesidad de tener comunicaciones constantes, ya que a cada procesador se le asignan tareas relativamente grandes. Por ejemplo, en el caso del algoritmo evolutivo al implementar la técnica de paralelismo de grano grueso, cada procesador puede iniciar el algoritmo con poblaciones distintas y así explorar diferentes zonas del espacio de búsqueda, comunicándose eventualmente. En este tipo de paralelismo todos los procesos tienen el mismo programa. No obstante, cada proceso trabaja con datos diferentes [Zhang, 2012], tal como ocurre con el algoritmo desarrollado, PEA-WDND.

El paralelismo de grano fino es un tipo de paralelismo muy específico que se basa en fragmentar procesos y requiere sincronizaciones frecuentes entre los procesadores ya que la paralelización es a nivel de instrucciones. Este paralelismo consiste en asignar a diferentes procesos partes especificas del dominio físico. Es decir, se indica a los procesos que instrucciones de un algoritmo deben ejecutar. Este tipo de paralelismo consiste en dividir las variables del problema para que sean procesadas por diferentes procesos. Un ejemplo de este paralelismo puede verse, en el algoritmo evolutivo, cuando diferentes procesos exploran de manera paralela diferentes partes del espacio de búsqueda, gracias a que cada uno de ellos trabaja con un fragmento de la población. Cada proceso ejecuta las instrucciones que le corresponden y frecuentemente puede comunicarse e intercambiar información con otros procesos. Esto ayuda al proceso a obtener soluciones de mejor calidad. Otro ejemplo, de paralelismo de grano fino puede verse cuando se identifica la tarea de mayor costo computacional y se definen instrucciones de tal forma que cada procesador realice solamente ciertas instrucciones. Por ejemplo, si la tarea que consume más recursos computacionales se refiere a aplicar el operador de mutación sobre todos y cada uno de los individuos de la población, con esta técnica cada procesador aplicaría mutación a cierto número de individuos.

El paralelismo de grano fino, al igual que el paralelismo de grano grueso, tiene como objetivo principal disminuir el tiempo requerido para encontrar soluciones a la vez que se mejora la calidad de las mismas. Puede verificarse que se cumpla el objetivo mediante la comparación de la ejecución del algoritmo paralelo con la ejecución del algoritmo

secuencial, observando de los resultados el tiempo de ejecución y la calidad de las soluciones. El paralelismo de grano fino presenta diferencias importantes con el paralelismo de grano grueso, tanto a nivel de software como a nivel de hardware [Goedecker, 2001]. A nivel de software, existen diferencias en el nivel de descomposición, en el paralelismo de grano fino una aplicación puede descomponerse a nivel de instrucciones mientras que en el paralelismo de grano grueso una aplicación se descompone a nivel de subrutinas o procedimientos. Así, este tipo de paralelismo provee un mayor grado de paralelización pero con comunicaciones constantes. Puede decirse que un programa paralelo de grano fino es lo contrario que un programa paralelo de grano grueso. Con base en las características del Algoritmo Evolutivo EA-WDND, en este trabajo el modelo de paralelización utilizado incluye la técnica de paralelización de grano grueso.

4.3. Modelos para el Diseño de Algoritmos Paralelos

Los modelos existentes para el diseño de algoritmos paralelos definen la forma en la que se realiza la asignación de trabajo entre los procesos que participan en la ejecución de un programa paralelo. La asignación de la carga de trabajo puede realizarse de forma estática, durante el tiempo de compilación del programa paralelo o bien de forma dinámica durante la ejecución de dicho programa. La asignación estática implica que las actividades de los procesos y las comunicaciones se definan desde la codificación del programa. En la asignación dinámica es necesario que los procesos se comuniquen constantemente para coordinar y realizar nuevas distribuciones de la carga de trabajo dependiendo de las necesidades que se presenten o de la disposición de los procesos, siempre con el objetivo de evitar tener procesos ociosos y minimizar el tiempo de cómputo requerido para la ejecución de una aplicación.

Los modelos más comunes para la paralelización de algoritmos son el modelo Maestro-Esclavos y el modelo de Islas, ambos con características definidas, ventajas y desventajas. En este trabajo de tesis se describe también un modelo propio propuesto en este trabajo de tesis. El modelo es llamado Maestro-Alumnos y es utilizado para la paralelización del algoritmo evolutivo desarrollado en este proyecto doctoral.

4.3.1. Modelo Maestro-Esclavos

El modelo maestro-esclavos, por sus siglas en inglés Master/Slaves, es un paradigma de paralelización que define el diseño de un algoritmo paralelo. En este modelo hay dos principales elementos: maestro y esclavos. El maestro es un proceso que controla la ejecución del programa paralelo y asigna trabajo a los procesos esclavos (procesos que realizan las tareas asignadas). Es responsable de la inicialización, la coordinación, los tiempos y las operaciones de salida [Rauber, 2010]. El proceso maestro maneja un flujo de alto nivel, con la finalidad de reducir las comunicaciones y cada proceso esclavo toma el control de las operaciones locales. Por ejemplo, en algoritmos evolutivos, un procesador maestro centraliza la población y administra la selección, los reemplazos de los individuos y el envío de las subpoblaciones a los esclavos para que éstos ejecuten ciertos operadores del algoritmo. Una vez realizadas sus tareas, los esclavos envían al maestro sus resultados [Baños, 2006]. Las acciones que realizan el maestro y los esclavos generalmente son idénticas, la diferencia es que éstas se realizan sobre datos diferentes [Wilkinson, 2008]. En [Coello, 2010] las evaluaciones de la función objetivo se distribuyen entre diversos procesos esclavos y el proceso maestro se encarga de ejecutar los operadores evolutivos. El proceso maestro puede dividir y distribuir a los individuos de una población entre los procesos esclavos disponibles. El proceso maestro, tiene también la tarea de controlar la ejecución de la función de evaluación y de almacenar los resultados que los procesos esclavos le envían.

El modelo maestro-esclavos es una estrategia fácil de visualizar desde la perspectiva de administración de algoritmos y también es relativamente fácil de implementar. Este modelo permite mantener la secuencia del algoritmo original. Es recomendable utilizar este modelo cuando el algoritmo presenta tareas con alto costo computacional como por ejemplo la generación y la evaluación de nuevos individuos. La desventaja principal del modelo maestro/esclavos que debe considerarse es que al tener un solo maestro que se encarga de la coordinación y administración de los procesos, los esclavos se comunican con un solo nodo ocasionando problemas de latencias por las comunicaciones frecuentes. Así mismo un proceso maestro puede permanecer ocioso durante un periodo de tiempo y muy saturado de actividades en otro periodo de tiempo, lo cual no es lo más recomendable.

4.3.2. Modelo de Islas

El modelo de islas es una estrategia de paralelización bio-inspirada [Cantu, 2000]. Es un paradigma de paralelización que define el diseño de un algoritmo paralelo. En este modelo el elemento principal es la isla representada por un proceso. Dentro de casa isla se tiene un conjunto de individuos que pueden migrar a otras islas. En la ejecución del algoritmo, el número de islas se encuentra definido generalmente con base en la infraestructura de hardware que se tenga disponible. Este modelo se basa en la idea intuitiva de dividir una población, del Algoritmo Evolutivo, en subpoblaciones para distribuirlas entre un conjunto de procesos disponibles [Cantú, 1999]. La finalidad de este modelo es explorar tanto como sea posible, diferentes áreas del espacio de búsqueda y mantener la diversidad de las poblaciones mediante el intercambio de los individuos a otras poblaciones [Alba, 2005] evitando así la convergencia prematura del algoritmo. El modelo de islas divide a la población original en un conjunto de subpoblaciones distribuidas entre diferentes procesos, los cuales son los responsables de la administración de la subpoblación que tengan asignada. De esta forma, en cada proceso se ejecutan todos los pasos del algoritmo evolutivo y ocasionalmente se migran individuos de una isla a otra. Para la conexión y el intercambio de datos, entre islas, se han definido diferentes topologías [Rucínski, 2010]. Entre ellas se encuentran: anillo, mallas bidimensionales y tridimensionales, estrella, hipercubo. Incluso puede utilizarse también una topología de comunicación aleatoria en la cual una subpoblación envía, de forma aleatoria, individuos a otra subpoblación [Crainic, 2003]. En el modelo de islas se deben configurar diferentes parámetros [Skolicki, 2005a; Skolicki, 2005b]: número de subpoblaciones, tamaño de la subpoblación, frecuencia de intercambios de individuos, número de individuos a emigrar, criterio para seleccionar los migrantes y los individuos a reemplazar, y topología de comunicación. Estos parámetros han sido estudiados ampliamente porque influyen en la calidad de la búsqueda y la eficiencia del algoritmo. Sin embargo, se sabe que los valores óptimos de dichos parámetros dependen principalmente de la naturaleza del problema abordado.

En el modelo de islas, existen políticas de migración que definen el funcionamiento de un algoritmo evolutivo paralelo, determinando la forma en que se llevará acabo la comunicación. Concretamente, son 5 las políticas principales de migración.

1) **Número de emigrantes** (m), $m \in \{0, 1, 2, 3, \ldots\}$: se refiere al número de individuos para intercambiar en una isla. El valor '0' significa que no hay interacción entre las subpoblaciones (no se migran individuos).

2) **Frecuencia de migración** (r), $r \in \{0, 1, 2, 3, \ldots\}$: es el número de generaciones que transcurren en la ejecución del algoritmo evolutivo sin establecer comunicación entre islas para el envío de individuos.

3) **Envío de migrantes** (S): se refiere a los individuos que se envían de una isla a otra. Existen métodos de selección disponibles en la literatura que ayudan a definir cuáles individuos migrarán, por ejemplo selección aleatoria, selección elitista, selección por torneo, entre otras.

4) **Reemplazo de migrantes** (R): esta política define la integración de los migrantes en la nueva subpoblación. En ocasiones, consiste en reemplazar a los peores individuos de una isla por los mejores individuos de otra isla.

5) **Sincronización**. Indica el tipo de envío/recepción entre las islas. Éste puede ser bloqueante o no bloqueante (tal como lo define el estándar MPI, véase 4.6). La sincronización, también define en qué momento se deben integrar los migrantes a la nueva población, puede ser en cuanto llegan o en algún momento posterior.

Con la utilización de la migración, este modelo puede explotar las diferencias en las subpoblaciones. Esta variación representa una fuente de diversidad genética. Sin embargo, si un número de individuos emigran en cada generación, ocurre una mezcla global y se eliminan las diferencias locales. Por otro lado, si la migración es infrecuente, es probable que se produzca convergencia prematura en las subpoblaciones. La desventaja principal del modelo de islas es que tiene diferentes parámetros que deben sintonizarse.

4.3.3. Modelo Maestro-Alumnos

Los modelos Islas y maestro-esclavos presentan ventajas propias que pueden utilizarse en la paralelización de algoritmos. En ese trabajo doctoral se hace uso de los dos modelos para formar un solo modelo híbrido que integra las bondades de ambos modelos. Este modelo hibrido es llamado maestro-alumnos.

El modelo maestro-alumnos (M/A) surge con la idea de aprovechar los beneficios que ofrecen los modelos previamente descritos y al mismo tiempo resolver los problemas que éstos presentan como desventajas. El modelo maestro-alumnos es una idea novedosa de este trabajo doctoral, que representa un modelo del ámbito escolar, donde se tienen diferentes grupos y en cada uno de ellos un conjunto de alumnos y un maestro que establece reglas y dirige a los alumnos en el aprendizaje y en la solución de problemas. Así el modelo de paralelización propuesto y llamado maestro-alumnos consiste en la simulación del modelo de ámbito escolar para el diseño y la ejecución de algoritmos paralelos. En concreto, el maestro-alumnos se utiliza para el diseño y la ejecución del algoritmo paralelo PEA-WDND desarrollado en este trabajo.

El modelo maestro-alumnos toma del modelo maestro-esclavos la idea de tener un proceso coordinador y varios procesos que se encargan del trabajo. Pero en lugar de tener un solo proceso coordinador, en el modelo maestro-alumnos se han definido tantos maestros como grupos se creen, evitando problemas de cuellos de botella o de que falle la ejecución del algoritmo cuando un proceso falla. El modelo maestro-alumnos permite la creación de diversos grupos de alumnos, cada uno de ellos con su propio maestro que actúa como coordinador. Esto se traduce también en la posibilidad de hacer estudio exhaustivo de diferentes componentes del AE, puesto que cada alumno puede definir su propia estrategia de solución, tal como ocurre en el ámbito escolar, pero en este trabajo basada en la definición de algoritmos evolutivos. El modelo maestro-alumnos toma del modelo de Islas la idea de que los procesos puedan trabajan de manera independiente, pero enviando cada cierto tiempo información al proceso maestro. Así en el modelo maestro-alumnos una población, del Algoritmo Evolutivo, puede dividirse en subpoblaciones para distribuirlas entre un conjunto de procesos disponibles. En este

modelo los individuos de una población (que representan las soluciones del problema a resolver por los alumnos) emigran a otras poblaciones con cierta frecuencia. La frecuencia es definida directamente por el profesor de cada grupo y se refiere al periodo en el que el profesor espera recibir los resultados. La finalidad de este modelo es explorar tanto como sea posible, diferentes áreas del espacio de búsqueda y mantener la diversidad de las poblaciones mediante el intercambio de los individuos a otras poblaciones además de que los procesos alumnos pueden definir su propia estrategia para la creación de la población y la generación de los individuos. Los procesos alumnos, son los responsables de aplicar los operadores de selección, cruce y mutación y en el tiempo de entrega definido por el profesor, envían a este sus mejores soluciones.

El modelo maestro alumnos presenta la ventaja también de ser tolerante a fallos. Esto significa, que incluso si a algunos nodos (alumnos) fallan no se afecta todo el sistema, porque el nodo maestro continúa su ejecución con los resultados de los nodos activos.Incluso, si un nodo maestro falla, la ejecución del algoritmo evolutivo paralelo PEA-WDND puede continuar y finalizar con éxito en los demás grupos definidos. Otra de las bondades de este modelo es que se evitan cuellos de botella puesto que los alumnos realizan sus propias actividades y se establecen comunicaciones eventuales. Finalmente, una ventaja más es que el modelo no requiere rutinas sofisticadas de sincronización porque éste aprovecha las ventajas de ejecutar el algoritmo paralelo en un ambiente grid homogéneo.

Tabla 4.4 Modelos de Paralelización

Modelo	Descripción	Ventajas	Desventajas
M/S	Modelo que tiene un proceso coordinador y varios procesos para la ejecución de tareas.	Relativamente fácil de visualizar e implementar.	En sistemas con muchos procesadores el maestro llega a ser cuello de botella dejando a los esclavos en espera. Si el maestro falla, todo el sistema falla.
Islas	Consiste en la evolución de subpoblaciones independientes que ocasionalmente intercambian individuos mediante un proceso de migración.	Diversidad genética mediante migración de individuos de una población a otra.	Introduce diversos parámetros que deben sintonizarse mediante análisis de sensibilidad.
M/A	Grupos de procesos que fungen como alumnos y un proceso que desempeña el papel de maestro.	Relativamente fácil de visualizar e implementar. Diversidad genética mediante migración de individuos de una población a otra. Diversos grupos de alumnos, cada uno con su propio maestro que actúa como coordinador. Mayor exploración de AE. Aprovecha el incremento de procesadores haciendo nuevos grupos independientes. No se afecta el sistema si fallan algunos nodos. Sincronización natural del PEA-WDND gracias al ambiente homogéneo.	Introduce parámetros que deben sintonizarse mediante análisis de sensibilidad. No recomendable para implementarse en ambientes heterogéneos a menos que se implementen nuevas técnicas de sincronización.

4.4. Arquitecturas de Cómputo Paralelo

Las plataformas de cómputo paralelo son entornos que permiten la implementación de algoritmos paralelos. Existen diferentes tipos de plataformas, algunas de ellas se conocen como plataformas de cómputo de alto rendimiento (HPC, High Performance Computing). Estas plataformas incluyen arquitecturas de memoria compartida (utilizada en los sistemas multiprocesadores simétricos, Symmetric Multi-Processing) y arquitecturas de memoria distribuida, propia de las plataformas de cómputo distribuidas (computación en clúster o computación Grid).

En los diseños antiguos de computadoras, los procesadores estaban conectados a la memoria global única y el acceso a ésta era a través de un canal común. Una ventaja de la memoria compartida es que los procesos pueden intercambiar datos simultáneamente. No obstante, la desventaja que presentan estas arquitecturas es que diversos procesos compiten por el acceso a la memoria, haciendo que el proceso de lectura y/o escritura sea lento, además de necesitar técnicas de sincronización. La sincronización es un concepto altamente importante en este tipo de sistemas. Compartir recursos globales implica riesgos. Por ejemplo, si dos procesos hacen uso al mismo tiempo de la misma variable global y ambos llevan a cabo tanto lecturas como escrituras sobre la variable, el orden en el que se ejecuten las lecturas y las escrituras es crítico. Así surge el concepto de exclusión mutua, también conocida como acceso único. Dos o más procesos no deben acceder a determinados recursos al mismo tiempo. Algunas técnicas de sincronización pueden implementarse a nivel hardware, a través de instrucciones máquina especiales o de inhabilitación por interrupciones. Otras técnicas se implementan a nivel de sistema operativo, con la utilización de diferentes técnicas de sincronización, algunas de ellas conocidas con el nombre de semáforos [Thakur, 2005b].

En arquitecturas de memoria compartida existe paralelismo implícito realizado automáticamente por el compilador pero en general, este tipo de paralelismo no es óptimo [Goedecker, 2001]. Por otra parte, las arquitecturas de memoria distribuida, conocidas también como plataformas de procesamiento paralelo masivo (MPP, Massive Parallel Processing) se utilizan para cálculos científicos de gran escala, con la finalidad de reducir

el tiempo de ejecución de las aplicaciones, mediante la implementación de aplicaciones paralelas. En estas arquitecturas la memoria global de las máquinas está formada por la memoria local de cada computadora, conectados mediante topologías de red punto a punto, hipercubo, entre otras. Algunos ejemplos destacados de las arquitecturas de memoria distribuida son la computación clúster y la computación Grid, las cuales se utilizan para la ejecución de aplicaciones paralelas de alto rendimiento y de alta productividad respectivamente.

Un clúster es un conjunto de computadoras, llamadas también nodos, interconectadas mediante una red de alta velocidad que trabajan de forma conjunta para la solución de un problema. A diferencia de una red tradicional, un clúster de computadoras de alto rendimiento tiene configuraciones especiales que habilitan la programación paralela, mediante el uso de bibliotecas de paso de mensajes. El concepto de clúster surge hace aproximadamente tres décadas. Los primeros sistemas de clúster fueron llamados “la supercomputadora del hombre pobre” debido a que, en los centros académicos, los investigadores solían dejar ejecutando las aplicaciones durante toda la noche en lugar de adquirir una supercomputadora [Becker, 2010]. A principios de los años 90 los clúster de computadoras estaban formados por agrupaciones de computadoras de escritorio y a finales de la misma década, en los clúster, se incluían también elementos como supercomputadoras.

Actualmente existen clúster de computadoras de diferentes tipos, de acuerdo con sus características [Rajkumar, 1999]. Existen clasificaciones con base en: los componentes, el uso de los nodos, el tamaño del clúster, el tipo de aplicaciones, entre otros. De acuerdo con los componentes que constituyen a un clúster éste puede clasificarse en homogéneo o heterogéneo. En los clúster homogéneos los componentes tienen características similares de velocidades de procesamiento, sistema operativo entre otros, mientras que en los clúster heterogéneos no existe tal similitud. De acuerdo con el uso de los nodos puede decirse que existen clúster dedicados, cuando éste es utilizado por un único usuario a la vez y no dedicado cuando éste es utilizado por varios usuarios a la vez. De acuerdo con las aplicaciones que se ejecuten en ellos se encuentran los clúster de alta disponibilidad y clúster de alto rendimiento, entre otros. La característica de los clúster de alta disponibilidad es el monitoreo de los servicios que están en uso, así como

también la sustitución de una máquina por otra cuando se presenten fallos en alguno de sus servicios, de tal forma que los recursos se encuentren disponibles para el procesamiento de una aplicación. El clúster de alto rendimiento tiene como característica principal la alta capacidad para procesar grandes volúmenes de información. En estos clúster es importante el tiempo de respuesta. De acuerdo con el tamaño de los clúster, éstos pueden pequeños (decenas) o grandes (miles) de computadoras y pueden pertenecer a un departamento, a una organización, e incluso puede formar parte de una Grid, Figura 4.6.

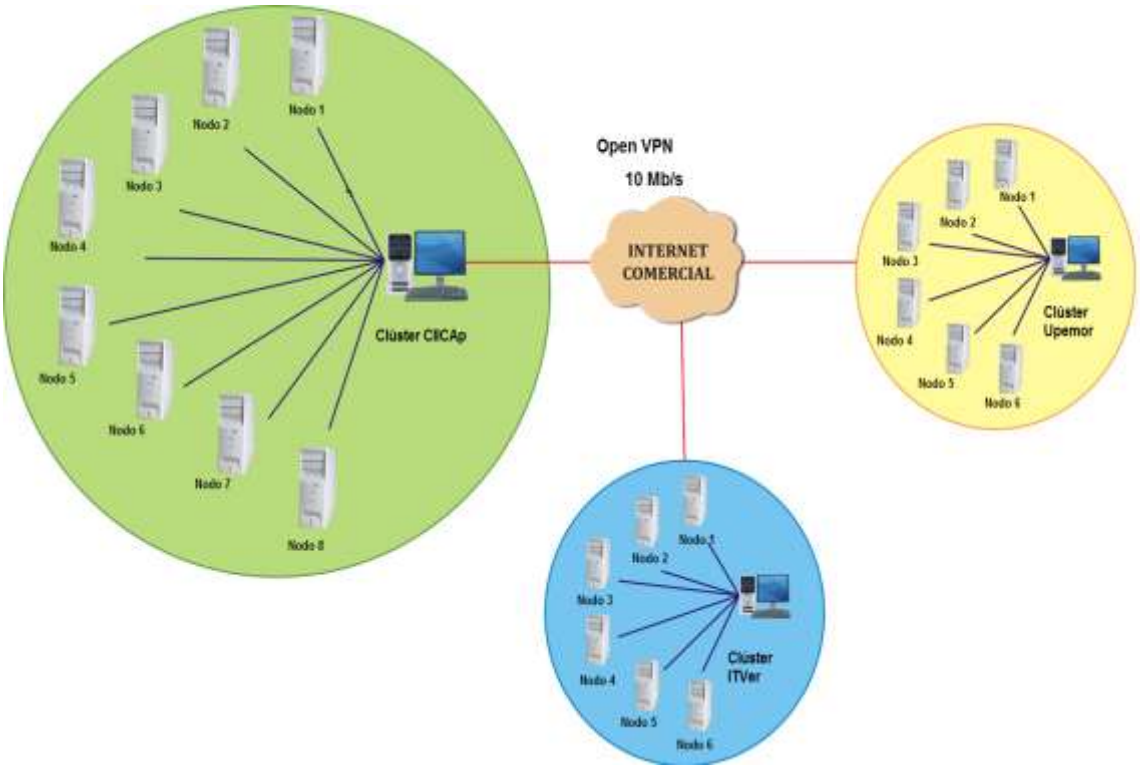


Figura 4.6 Tres Clúster unidos a través de una red Privada Virtual

En general, el beneficio principal que ofrecen los clúster de computadoras es el uso de los recursos a bajo costo. Una de las ventajas de los clúster es su administración, la cual es relativamente sencilla cuando se trata de clúster de computadoras dedicado. Sin embargo, en clúster de computadoras heterogéneo se tiene el problema de que existen máquinas con distintas características y capacidades, lo cual en algún momento puede hacer que las aplicaciones paralelas sean más lentas. Otra de las ventajas de los clúster de computadoras es que éstos son “fáciles” de escalar y a un bajo costo. Es decir, se

pueden añadir nuevos equipos para contar con mayor poder de cómputo. Así mismo, el mantenimiento es relativamente sencillo.

En la Universidad Autónoma del Estado de Morelos se hizo la instalación y la configuración de un clúster de alto rendimiento, el cual es utilizado para realizar pruebas experimentales en este trabajo de tesis. El clúster fue desarrollado, utilizando el sistema operativo Linux Red Hat edición empresarial [Cruz, 2010]. El primer paso en el desarrollo del clúster es la configuración de la red privada para posteriormente hacer una serie de configuraciones que permiten trabajar cómputo paralelo. Algunos de estos servicios incluyen la instalación de un sistema de archivos (Network File Systems, NFS), la instalación y configuración de entornos que permiten trabajar con paso de mensajes en la programación paralela, la configuración de herramientas para planeación de la ejecución de los trabajos así como la administración y monitoreo de la operación del clúster. Este clúster fue utilizado para la ejecución de los algoritmos desarrollados en sus primeras versiones, con plataformas de 32 bits y en las versiones finales del algoritmo evolutivo se utilizaron plataformas de 64 bits.

Además del clúster de computadoras se utilizó también una plataforma llamada Grid (constituida por clúster de clúster de computadoras, geográficamente distribuidas) para para la implementación del algoritmo paralelo PEA-WDND.

El término Grid surge a principios de los años 90 para hacer referencia a una infraestructura de cómputo distribuida para aplicaciones científicas y de ingeniería avanzadas [Berman, 2003]. Este término se debe a una analogía con la red de energía eléctrica porque proporciona un acceso constante, ininterrumpido y de forma transparente al usuario, al cual no le interesa identificar a la fuente que le proporciona los servicios sino únicamente necesita hacer uso de ella. La Grid, surge como un proyecto para conectar supercomputadoras que se encontraban dispersas en diferentes lugares geográficos, de tal forma que el usuario pudiera acceder a los recursos computacionales de forma transparente. El término “transparente”, en sistemas distribuidos, se refiere a que el usuario no necesita saber la ubicación de los recursos que le ofrecen un servicio, sólo requiere que se encuentren disponibles para atender sus peticiones en cualquier momento. La Grid tiene sus raíces en la e-Science y ha evolucionado a cómputo paralelo, distribuido y de alto rendimiento [Nokolaos, 2011]. Los principales beneficios de utilizar

esta plataforma incluyen: 1) mejorar el uso de los recursos, 2) permitir la ejecución de aplicaciones a gran escala que no pueden ser ejecutadas en una sola computadora y 3) utilizar recursos de diferentes ubicaciones y dominios, entre otros [Bellatreche, 2011]. Los precursores de la computación Grid [Foster, 2002a; Foster, 2002b], definen a la Grid como un enfoque en el que existen organizaciones virtuales unidas para lograr objetivos comunes. Algunos ejemplos de organizaciones virtuales son proveedores de servicio de almacenamiento, equipos de trabajo empresarial, universidades que trabajan en colaboración en proyectos de investigación, entre otras. La computación Grid ha sido motivada por la necesidad de contar con una infraestructura de cómputo de alto rendimiento, formada por la cooperación de diversas instituciones para tener acceso a los recursos computacionales a bajo costo. Es importante mencionar que cada organización define sus propias reglas de operación y acceso a los recursos. La Grid coordina los recursos a través de protocolos abiertos, de propósito general, para proporcionar al usuario la impresión de trabajar con un sistema único cuando en realidad existe un conjunto de computadoras que unidas proporcionan al usuario un sistema potente en cuanto a recursos de memoria y procesamiento.

Actualmente, existe un proyecto llamado Mini Grid Morelos [Cruz, 2012b], que consiste en una Grid interinstitucional para desarrollar proyectos de investigación en e-Ciencia. La Grid Morelos [Cruz, 2012b] es la primera Grid de cómputo intensivo en el país y se encuentra formada por tres clúster de alto rendimiento, localizados en tres instituciones educativas. El clúster “Cuexcomate” que se encuentra dentro del laboratorio de Optimización y Software y pertenece al área de Ingeniería y Ciencias Aplicadas de la Universidad Autónoma del Estado de Morelos. El clúster “Texcal” que se encuentra en el laboratorio de Informática dentro de la Universidad Politécnica del Estado de Morelos. Finalmente, el clúster “Nopal” que está en el área de Informática en el Instituto Tecnológico de Veracruz, Figura 4.7. Los clúster Cuexcomate, Texcal y Nopal se encuentran enlazados a través de Internet 2 para habilitar la ejecución de aplicaciones paralelas distribuidas uniformemente en la Grid [Cruz, 2010b]. A diferencia de los clúster, la Grid se enfoca principalmente en la construcción de aplicaciones científicas de propósito general. Una de las ventajas de la Grid es que ofrece mecanismos para compartir recursos de forma eficiente, a bajo costo [Chakrabarti, 2007], mediante la unión de múltiples clúster de computadoras. Sin embargo, una de las desventajas que presenta

la Grid y que debe considerarse es que el acceso a los recursos depende directamente de las reglas que defina cada organización, por lo que los recursos pueden aparecer o desaparecer en cualquier momento sin previo aviso para los usuarios. Incluso la grid completa o parte de ella puede estar disponible o sin servicio lo cual debe considerarse para la ejecución de los algoritmos paralelos.

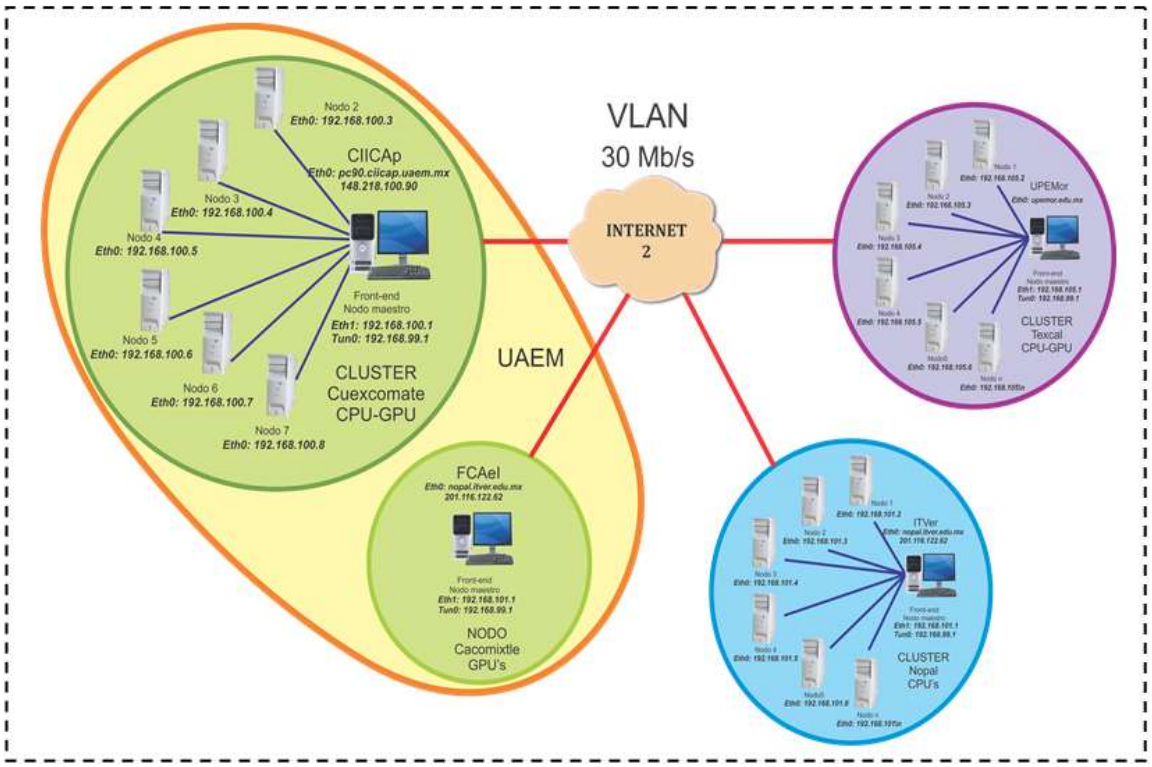


Figura 4.7 Esquema de una Grid formada por 3 Instituciones

La Grid Morelos se utilizó para la implementación del algoritmo PEA-WDND, desarrollado en este trabajo, porque permite la ejecución de un programa computacional en todos los clúster que la componen. Para el diseño del algoritmo PEA-WDND se utilizó la técnica de paso de mensajes, la cual es una de las características propias de las arquitecturas de memoria distribuida, que permite establecer comunicación entre los procesos durante la ejecución de las aplicaciones paralelas.

4.5. Métricas para el rendimiento de Algoritmos Paralelos

El objetivo de cómputo paralelo es reducir el tiempo de cómputo empleado por un algoritmo para la solución de un problema. En la actualidad, existen medidas de rendimiento de algoritmos paralelos que permiten determinar la eficiencia de un algoritmo. La métrica más comúnmente utilizada por la comunidad científica es conocida como *speedup* y se refiere a comparar la ganancia de velocidad que se obtiene con la ejecución de un algoritmo paralelo con respecto a un algoritmo secuencial. El *speedup* (S) se define como:

$$S(np) = \frac{T(1)}{T(np)},$$

donde T es el tiempo de ejecución y *np* es el número de procesos o núcleos disponibles en una máquina paralela. Así, T(1) se refiere a la ejecución del algoritmo en un solo proceso y T(*np*) se refiere a la ejecución del algoritmo utilizando un número procesos *np*, donde *np* > 1.

La ganancia de velocidad ideal es una función lineal que se refiere a que la ejecución de un algoritmo paralelo requiere un tiempo *np* veces menor que el tiempo requerido por un solo procesador para la solución de un problema. En la práctica no es muy común encontrar la ganancia de velocidad ideal, si se considera que la ejecución de un algoritmo paralelo implica tener factores como latencia y tiempos de comunicaciones entre procesos. No obstante, existen referencias como [Alba, 2002] en las que se muestra una ganancia de velocidad superlineal. Es decir, esta ganancia de aceleración del algoritmo se encuentra por encima del número de procesadores: $S(np) = \frac{T(1)}{T(np)} > np > 1$.

De acuerdo con [Baños, 2010], algunos factores que motivan la superlinealidad son:

- Incremento de los recursos en los sistemas paralelos. El incremento de recursos tales como memoria principal, memoria cache y número de procesadores permite que los programas paralelos puedan explotar los recursos y obtener valores de ganancia de velocidad superiores a los valores lineales.

- Características propias de los algoritmos paralelos. Algunos algoritmos paralelos evolucionan de diferente forma dependiendo del número de procesadores que utilicen y dependiendo principalmente del modelo de paralelización utilizado.

- Ineficiencia de los algoritmos secuenciales. En numerosas ocasiones se realizan implementaciones secuenciales que no son del todo óptimas. Esto conlleva a que dichas implementaciones requieran de tiempos de ejecución elevados para obtener el resultado. Al hacer uso adecuado del procesamiento paralelo un algoritmo secuencial puede obtener una ganancia de velocidad de superlineal.

4.6. Estándar MPI

El estándar MPI (*Message Passing Interface*) [MPI, 2013] es una biblioteca de funciones que se basa en la técnica de paso de mensajes para habilitar la computación paralela, permitiendo la codificación, la ejecución y la comunicación de procesos en aplicaciones paralelas. Este estándar fue desarrollado por un foro en el que participaron universidades, laboratorios y empresas. Su diseño se realizó inspirado en las máquinas de arquitectura distribuida, en donde cada procesador tiene su propia memoria local y el intercambio de información se hace a través del envío de mensajes. MPICH [MPICH, 2013] es sin duda una implementación importante de MPI. Existen implementaciones comerciales desarrolladas por las grandes compañías de computadores paralelos que se basan en ella. La primera versión de MPICH fue escrita durante el proceso de estandarización de MPI. De hecho, las experiencias de los autores de MPICH se convirtieron en una gran ayuda para el Foro MPI en el proceso de desarrollo del estándar.

El estándar MPI ha sido probado, aceptado y utilizado ampliamente por la comunidad científica [Goedecker, 2001], ya que es de libre acceso y distribución, y puede utilizarse desde un programa, en lenguajes de programación como C, Fortran, entre otros.

MPI, permite realizar aplicaciones en las que todos los procesos ejecutan el mismo programa, pero no necesariamente las mismas instrucciones, e incluso pueden ejecutar

las instrucciones en diferentes tiempos o bien trabajar sobre un conjunto distinto de datos. La biblioteca MPI provee el acceso a un conjunto de funciones que permiten la ejecución de un programa distribuido, lo cual no significa que un trabajo se deba ejecutar en múltiples máquinas, de hecho, es posible ejecutar diversos procesos en una sola computadora [Landman, 2008]. Esta biblioteca está formada por un conjunto de primitivas que permiten el paso de mensajes entre procesadores conectados en una red. MPI asume que trabajará con procesos que no comparten espacios de memoria, de tal forma que si un proceso cambia el valor de una variable, los demás procesos no observan el cambio, a menos que se envíe la información de forma explícita mediante el paso de mensajes. Es necesario tener presente que el estándar MPI se encuentra disponible como una biblioteca, así que si se desea hacer uso de las primitivas que contiene el estándar es necesario incluirla en un programa. Todo programa que hace uso de esta biblioteca tiene una estructura similar, que puede variar dependiendo del lenguaje de programación que se utilice. En este trabajo se hace uso de lenguaje C, y la estructura del programa es similar a la Figura 4.8.

```
1.  #include <mpi.h>

2.  int main (int argc, char **argv)
3.  {
4.   int NumProc, IdProc;
5.   MPI_Init (&argc, &argv);
6.   MPI_Comm_size(MPI_COMM_WORLD, &NumProcs);
7.   MPI_Comm_rank(MPI_COMM_WORLD, &IdProc);
8.   /* CUERPO DEL PROGRAMA*/
9.   MPI_Finalize();
10. }
```

Figura 4.8 Estructura de un Programa MPI

La línea 1 del programa de la figura 4.8, sirve para incluir la biblioteca de mensajes MPI y así hacer uso de las funciones que ésta contiene. La línea 5 permite inicializar el entorno de trabajo para poder utilizar los comandos y las rutinas definidas en MPI. La línea 6 obtiene el número de procesos disponibles que pueden participar en la ejecución de una aplicación paralela. La línea 7 se refiere al identificador, único y de tipo entero, de cada uno de los procesos disponibles (*IdProc*). Este identificador, es útil para tener un control de los procesos cuando se ejecuta un programa. A través del identificador es posible referirse a un proceso de forma explícita, ya sea para asignar actividades, para

enviar o recibir datos o simplemente para reconocer la identidad de un proceso. Los identificadores se asignan automáticamente, a todos los procesos involucrados en la ejecución del programa paralelo, cuando se inicializa el entorno MPI. En la Figura 4.9 se observa que la asignación de los identificadores se hace de forma contigua e inicia en el número 0. Se observa también al comando *MPI_COMM_WORLD*. Este, es el comunicador por defecto en MPI que se utiliza en todas las comunicaciones, ya sean punto a punto o colectivas. Es el comunicador universal o global que determina que todos los procesos involucrados en la ejecución de la aplicación pueden comunicarse. Es decir, el comunicador representa un dominio de comunicación que define al conjunto de procesos que intercambian mensajes e indica cuáles son los procesos que participan en las comunicaciones.

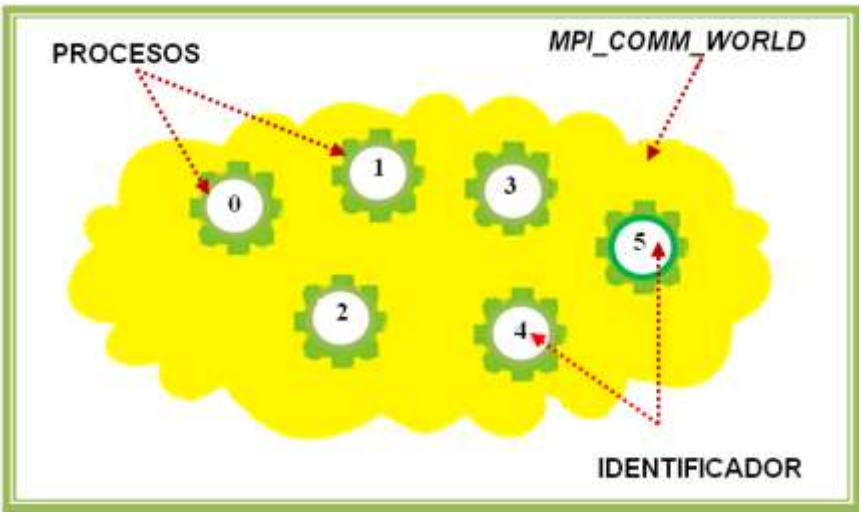


Figura 4.9 Elementos básicos para la comunicación en MPI

La línea 8 se refiere al cuerpo de programa, el cual es similar al cuerpo de un programa escrito en lenguaje C pero incluye también comandos específicos de MPI. Aquí se colocan todas las instrucciones para desarrollar la funcionalidad del programa. Finalmente, la línea 9 sirve para finalizar el entorno, y liberar la memoria una vez que se terminan de utilizar los comandos y las rutinas de MPI. La tabla 4.3 muestra algunos de los comandos más comunes de MPI, mismos que pueden reconocerse porque van precedidos por el prefijo MPI. Las instrucciones más representativas de MPI son *MPI_Send* y *MPI_Recv* que se utilizan para el envío y recepción de datos respectivamente.

En [MPI, 2013] puede encontrarse información detallada sobre las características y sintaxis de los comandos y rutinas de MPI. La biblioteca MPI tiene definidas rutinas para diferentes tipos de comunicación, dependiendo de lo que se necesite. Así, al utilizar el estándar MPI las comunicaciones pueden elegirse con base en el tiempo que los procesos permanecen bloqueados tras el envío o recepción de mensajes. Las comunicaciones pueden ser bloqueantes o no bloqueantes.

Tabla 4.5 Comandos comunes de MPI

FUNCIÓN	DESCRIPCIÓN
MPI_Init	Inicializa el entorno de MPI
MPI_Finalize	Finaliza el entorno de MPI
MPI_Bcast	Envía datos a todos los procesadores
MPI_Gather	Obtiene datos de todos los procesadores
MPI_Scatter	Distribuye los datos a todos los procesadores
MPI_Comm_rank	Obtiene los identificadores de los procesadores
MPI_Comm_size	Obtiene el número de procesadores en ejecución
MPI_Send	Envía Datos en un mensaje
MPI_ Recv	Recibe Datos en un mensaje
MPI_Wtime	Devuelve el tiempo en segundos
MPI_Get_processor_name	Devuelve el nombre del procesador actual

Las comunicaciones bloqueantes son aquellas en las que los procesos que envían o esperan recibir algún mensaje permanecen ociosos hasta que se completa la operación. Es decir, cuando un proceso hace uso de una función de comunicación, dicho proceso permanece bloqueado hasta que la operación finaliza. En el caso de la recepción, la operación finaliza cuando el mensaje enviado se encuentra en memoria listo para ser utilizado. En caso de la emisión la operación puede definirse como finalizada cuando el receptor ya tiene el mensaje. Este tipo de comunicación puede utilizarse cuando es necesario asegurarse de que un receptor recibe el mensaje antes de que ambos puedan continuar con sus actividades. Una analogía de esta comunicación puede verse cuando se habla por teléfono. Es necesario que un elemento, ya sea el emisor o el receptor, permanezca escuchando antes de poder responder para evitar problemas de comunicación. En este esquema se observa una sincronización en los tiempos de espera para asegurar que la comunicación sea adecuada.

En las comunicaciones no bloqueantes los procesos que envían o esperan recibir algún mensaje pueden continuar realizando otras actividades, ya que suponen que el sistema se encargará de realizar la operación. Sin embargo, cuando se utilizan estas comunicaciones es necesario averiguar posteriormente si la información ha llegado al destinatario. Un ejemplo de este esquema pudiera verse en el envío de un correo electrónico. Un emisor envía el mensaje e inmediatamente continúa realizando otras actividades, sin tener que esperar a que el emisor reciba el mensaje.

Otra clasificación importante de las comunicaciones en MPI, se determina con base en el número de procesos que intervienen en la comunicación. En general, la comunicación, puede ser de dos tipos 1) punto a punto y 2) colectiva.

4.5.1. Comunicación Punto a Punto

La comunicación punto a punto es el tipo de comunicación más simple que existe, en MPI, para la transferencia de datos entre un par de procesos [Ropo, 2009]. La comunicación punto a punto consiste en tener exactamente un proceso emisor que ejecuta una operación de envío y un proceso receptor que ejecuta una operación de recepción, Figura 4.10.

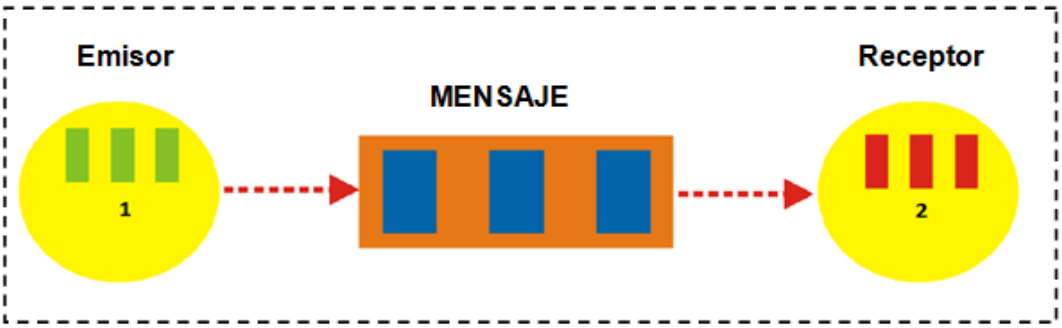


Figura 4.10 Comunicación Punto a Punto

Las comunicaciones punto a punto son útiles cuando se utilizan en modelos de comunicación irregular. Por ejemplo, en un modelo en el que cada proceso intercambia regiones de datos con otros procesos esporádicamente. Otro ejemplo es cuando un proceso se encarga de enviar a los demás procesos nuevas tareas a realizar, cada cierto tiempo, o cada vez que completan una actividad [Nikolaevskay, 2012]. La comunicación

punto a punto bloqueante debe definirse de forma explícita en la codificación de un programa paralelo. Generalmente la comunicación se define a través de peticiones entre los procesos, eventos que ocurren en el programa o simplemente en intervalos de tiempo específicos. Cuando se utiliza este tipo de comunicación, tanto el emisor como el receptor esperan a que el mensaje se transfiera completamente antes de poder continuar. La comunicación punto a punto bloqueante garantiza que cada mensaje llegue su destino íntegro y sin errores. Sin embargo, es necesario tener cuidado de que cada mensaje de envío tenga su correspondiente mensaje de recepción. Es decir, si se envían 10 mensajes (a un proceso o más procesos) debe haber 10 receptores que los reciban, de otra forma pueden tenerse problemas comunes conocidos como abrazo mortal (*deadlock)*[Chandra, 2010], los cuales son difíciles de identificar, aunque de antemano se sabe que ocurren cuando un envío no ha sido recibido por el receptor, o bien cuando un receptor se queda esperando un envío por tiempo indefinido. Así una aplicación puede terminar sin éxito cuando no se observa progreso durante cierto periodo de tiempo, generalmente minutos debido a que un proceso se queda esperando algo que nunca ocurrirá.

Es recomendable que las operaciones de envío y recepción sean del mismo tipo. Es decir, si se utiliza un envío no bloqueante debe hacerse la recepción no bloqueante también. Así mismo, se debe verificar que los datos que sean del mismo tipo cuando se envían y cuando se reciben. Adicionalmente, se debe tener cuidado de reservar suficiente espacio en memoria para la recepción de los datos, ya que si el espacio no es suficiente el mensaje no podrá almacenarse completo. Para el envío y recepción bloqueantes se utiliza la instrucción *MPI_Send* y *MPI_Recv*, respectivamente. Una de las ventajas de las comunicaciones punto a punto bloqueantes es la facilidad para ser implementada. No obstante, es importante definir el tipo de comunicación para evitar que los procesos estén ociosos y con ello retrasen la ejecución total del programa paralelo. El algoritmo PEA-WDND, desarrollado en este trabajo de tesis, utiliza la comunicación punto a punto bloqueante previamente descrita. Con este tipo de comunicación el algoritmo tiene éxito en el envío y recepción de mensajes. Esto se debe, en gran medida al modelo de paralelización implementado. Con el modelo maestro-alumnos, un maestro puede saber el número de alumnos que le enviarán trabajos y se prepara para recibirlos.

4.5.2. Comunicación Colectiva

MPI ofrece diferentes modelos de comunicaciones colectivas. Algunas involucran únicamente los datos y otras involucran tanto las operaciones como las comunicaciones. La comunicación colectiva, es una extensión natural del paradigma de paso de mensajes. En esta comunicación todos los procesos de un grupo están involucrados en el intercambio de datos de forma simultánea.

Las primeras versiones de MPI que ofrecían las comunicaciones colectivas no funcionaban muy bien, ya que MPI se enfocaba principalmente en el funcionamiento de las comunicaciones punto a punto. Derivado de esto, algunos programadores implementaban sus propias versiones de comunicación colectiva mejorando considerablemente los resultados obtenidos con la comunicación punto a punto. Gracias a décadas de investigación, las versiones recientes de MPI ofrecen rutinas de comunicación colectiva que ayudan a los algoritmos a ser más optimizados y a tener un mejor funcionamiento que cuando el programador diseñaba sus propias rutinas de comunicación colectiva [Ada, 2010]. No obstante, las comunicaciones colectivas continúan siendo un área activa de investigación. MPI permite agregar modularidad a la ejecución de un programa mediante la definición de grupos, lo cual es una de las principales ventajas de la comunicación colectiva que se aprovecha en este trabajo de tesis. El objetivo de crear grupos es reducir tanto la latencia como el tránsito en la red. Un grupo es una colección de procesos, con alguna característica común, que pueden comunicarse con los procesos del mismo grupo o de otros grupos. Esto es posible gracias a que los procesos además de tener un identificador global, tienen también un identificador local que los identifica dentro del grupo. La Figura 4.11 es una representación de la implementación de grupos para el algoritmo PEA-WDND. En esta figura se muestran tres grupos, con las letras G1, G2 y G3. Todos los procesos del grupo tienen un identificador global que va desde 0 hasta *n*-1, siendo n el número de procesos existentes. En la Figura puede observarse que los procesos se etiquetan con P0, P1,…, hasta P11 lo cual indica que los 12 procesos definidos pertenecen a un grupo y pueden comunicarse a través del comunicador global, *MPI_COMM_WORLD*. MPI permite la creación de comunicadores nuevos dentro del comunicador global. En cada uno de los grupos se define un nuevo comunicador (*MPI_COMMG1, MPI_COMMG2, MPI_COMMG3*) para lograr que los procesos se

comuniquen, utilizando operadores de comunicación colectiva o de comunicación punto a punto como el caso del algoritmo PEA-WDND. Es importante decir que en la implementación del algoritmo de este trabajo la comunicación se establece exclusivamente con los procesos que se encuentran dentro de un grupo específico. Así el sistema de MPI, al definir el grupo le asigna a cada uno de los procesos un nuevo identificador local. Puede decirse que los procesos tienen ahora un identificador global y un identificador local en el grupo al que pertenecen. La asignación de identificadores locales se hace de la misma forma que la asignación de identificadores globales. Si un comunicador contiene *n* procesos, los identificadores locales se asignarán como números enteros de 0 a *n*-1; por lo que en el grupo G1 el proceso P0 tiene identificador global 0 e identificador local 0. En este caso coinciden ambos identificadores. Pero haciendo referencia al grupo G2 el proceso P8 tiene identificador global 8 e identificador local 0, el proceso P9 tiene identificador global 9 e identificador local 1, y así sucesivamente. Puede observarse que los procesos conservan su identificador global pero se les asigna un nuevo identificador que inicia nuevamente en 0 y termina en tamaño del grupo menos 1, y sirve para comunicarse exclusivamente con los procesos que pertenecen al grupo.

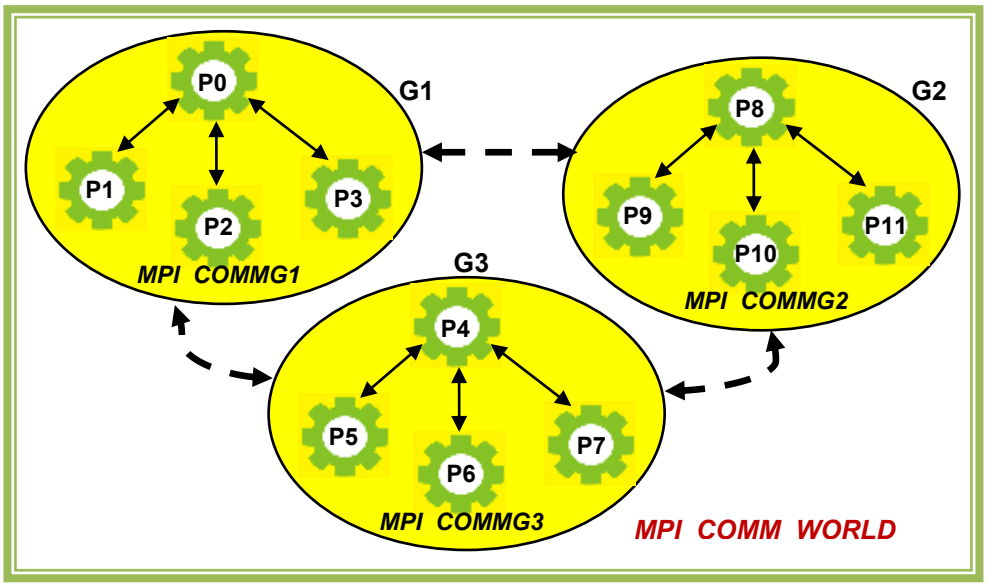


Figura 4.11 Grupos de Procesos

Las funciones típicas de la comunicación colectiva son *MPI_Bcast* y *MPI_Reduce*, que permiten enviar y recibir datos. *MPI_Bcast* obtiene los datos de un proceso y los envía a los demás procesos del grupo y por el contrario, *MPI_Reduce* obtiene los datos

de todos los procesos del grupo y los almacena en la memoria de un proceso. De acuerdo con [Pad, 2010], otras ventajas de implementar la comunicación colectiva son:

1. Simplicidad en la programación paralela y distribuida, sin necesidad de que el usuario debe implementar modelos complejos de comunicación mediante operaciones primitivas de comunicación.
2. Aumento en el nivel de abstracción cuando se implementan algoritmos paralelos, manteniendo ocultos los detalles de la comunicación colectiva del sistema en aplicaciones paralelas
3. Disminución en los costos de comunicación ya que el trabajo está distribuido en grupos y se evita que un solo proceso tenga sobrecarga de trabajo y comunicaciones constantes, para evitar problemas de latencia y ancho de banda en la red.

La latencia y el ancho de banda son factores que definen la capacidad y la velocidad de la red de computadoras. Estos factores están asociados con el funcionamiento de las aplicaciones ejecutadas en memoria distribuida, que afectan en gran medida la eficiencia de un programa paralelo. La latencia es el tiempo que se necesita para transmitir un mensaje de un nodo origen a un nodo destino. La latencia puede reducirse enviando menor cantidad de mensajes, aunque éstos sean de mayor tamaño para evitar que gran parte del tiempo de la ejecución de la aplicación paralela sea utilizado en las comunicaciones. La latencia es muy importante en procesamiento paralelo porque determina el tamaño granular, es decir, lo mínimo que debe tardar la ejecución de un segmento de código para que su ejecución en paralelo sea rentable, hablando en términos de rendimiento computacional. Básicamente si un segmento de código se ejecuta en menos tiempo del que se emplea en transmitir su resultado, entonces será más rápido ejecutar dicho segmento de código de manera secuencial en el procesador que necesita dicho resultado. De este modo la ejecución secuencial evitaría la sobrecarga en la comunicación. El ancho de banda se refiere a la velocidad máxima de la transferencia de un mensaje en megabits por segundo.

En este trabajo doctoral se ha considerado el concepto de la latencia y el ancho de banda que ofrece la plataforma grid en la que se trabaja. El algoritmo PEA-WDND tiene implementadas funciones que permiten mandar la información serializada en paquetes de

mayor tamaño, evitando comunicaciones frecuentes. El resultado de este procedimiento se ve reflejado en la eficiencia del algoritmo. El algoritmo PEA-WDND hace uso principalmente de la comunicación punto a punto, pero en algún momento usa también funciones de la comunicación colectiva. El algoritmo PEA-WDND utiliza la comunicación colectiva para dar a conocer a todos los procesos la lista de maestros con la cual cada alumno debe comunicarse, véase función principal (*main*) en Apéndice II. Y se utiliza la comunicación punto a punto para el envío de tareas de los alumnos a los maestros.

La comunicación punto a punto y colectiva implementada en el algoritmo PEA-WDND es posible gracias al modelo de paralelización propio propuesto en este trabajo de tesis. La combinación de las técnicas de comunicación permiten tener un algoritmo novedoso llamado PEA-WDND, tanto por la plataforma Grid utilizada como por el modelo de paralelización, el cual es una hibridación de las técnicas existentes.

4.7. Módulo Epanet para Análisis Hidráulico de la Red

Epanet es un programa que fue desarrollado por la Agencia de Protección Ambiental de los Estados Unidos Americanos [Rossman, 2000], con la finalidad de entender y mejorar los flujos hidráulicos en una red de distribución de agua.

Epanet es una herramienta que permite realizar la simulación hidráulica de una red de distribución de agua. Esta herramienta contiene un conjunto de rutinas que permiten analizar diferentes tipos de redes de distribución de agua. Con Epanet es posible conocer los flujos que circulan en las tuberías de la red, las presiones que se alcanzan en los nodos de consumo, el comportamiento de la red cuando ésta utiliza bombas de potencia, la energía consumida por la red, el costo del consumo energético, entre otras. Cabe mencionar que no existe límite en cuanto al tamaño de la red que Epanet puede procesar. Así mismo, Epanet tiene diferentes fórmulas que pueden utilizarse para calcular las pérdidas de carga de la red, las más conocidas: Hazen Williams, Darcy-Weisbach.

Epanet al ser un software de dominio público puede ser copiado y distribuido libremente. Puede utilizarse en diferentes plataformas, como Windows, Linux, entre otras. También puede utilizarse en diferentes entornos de programación. Este software ha sido probado y utilizado con éxito para simular el comportamiento hidráulico de una red de

distribución de agua, por diversos investigadores [Eusuff and Lansey, 2003; Reca, 2006; López-Ibáñez, 2011]. Actualmente, existe un kit de herramientas de programación de Epanet (toolkit, conjunto de herramientas), el cual es una extensión del paquete de simulación Epanet que puede utilizarse en plataformas Windows como una biblioteca de enlace dinámica (DLL). Este kit de herramientas provee diferentes funciones que permiten a los programadores personalizar el uso de un módulo hidráulico en sus propias aplicaciones. El código fuente original de Epanet, escrito en lenguaje C, puede descargarse del sitio oficial [Epanet, 2015]. El kit de herramientas Epanet, está disponible como una biblioteca de funciones que puede ser llamada desde programas escritos en diferentes lenguajes de programación. La mayoría de los autores que abordan el problema de Diseño de Redes de Distribución de Agua, utilizan a Epanet en plataformas Windows [Reca, 2006, Baños, 2007, Baños 2010]. Consecuentemente, existen muy pocas referencias de investigadores que utilizan el conjunto de herramientas de Epanet en lenguajes Linux. [López-Ibáñez, 2008] utiliza el kit de herramientas en ambiente Linux, pero la utiliza como una versión propia modificada del Epanet original, que contiene adaptaciones para resolver su propio problema, entre otras tiene funciones para la distribución en una red de agua utilizando la técnica por bombeo.

En este trabajo se utiliza el kit de herramientas original como un módulo hidráulico para dar solución al modelo de satisfacción de restricciones.

La metodología de solución, para el problema abordado, consistió en el desarrollo de un módulo de optimización que trabaja en conjunto con el módulo hidráulico, en ambiente Linux. La unión de dichos módulos permite resolver los dos modelos matemáticos planteados en este trabajo, los cuales representan al problema de Diseño de Redes de Distribución de Agua.

A pesar de que Epanet es un software libre, existe poca documentación acerca del procedimiento a realizar para poder utilizar el kit de herramientas en ambiente Linux. En este trabajo, se ha realizado el procedimiento que se describe a continuación:

1. Descargar la biblioteca Epanet de la página oficial. Para descargar la biblioteca Epanet es necesario acceder a la página oficial de la Agencia de Protección Nacional Estadounidense [Epanet, 2015]. Existe Epanet para Linux para plataformas de 32 bits y para plataformas de 64 bits. El programador debe elegir la plataforma acorde a sus necesidades. Una vez elegida la versión de Epanet

adecuada, ésta debe ser descargada y descomprimida utilizando la instrucción: *tar -xzvf filename-epanet.tar.gz*

2. Adaptar el kit de herramientas de Epanet, llamado “toolkit.h” para que funcione en ambiente Linux. Para ello es necesario crear un script que permita generar una biblioteca estática, la cual contendrá la funcionalidad de Epanet. Para crear la biblioteca estática se debe realizar lo siguiente:
   - Compilar los archivos que se encuentran dentro de la descarga del código fuente de Epanet (archivos con extensión .c) para obtener los archivos objeto(archivos con extensión .obj)
   - Empacar los archivos objeto utilizando la instrucción: ***ar –rcs ../libstaticname.a filename.o***, donde “libstaticname”, en este trabajo es llamado epanetsl.a. Éste es el nombre de la biblioteca estática resultante de empacar los archivos objeto. Para tener toda la funcionalidad de la biblioteca Epanet, es necesario incluir en la biblioteca estática todos los archivos objeto de Epanet (epanet.o hash.o hydraul.o inpfile.o input1.o input2.o input3.o mempool.o output.o quality.o report.o rules.o smatrix.o). En este punto se crea una biblioteca estática con el nombre de epanetsl.a y ésta es colocada en el mismo directorio donde se encuentra el Algoritmo Evolutivo PEA-WDND, para evitar la configuración de variables de entorno.

3. Definir en el módulo de optimización, el modulo hidráulico para poder utilizarlo. Esto se realiza incluyendo en el programa de optimización (en este trabajo PEA-WDND) la biblioteca “toolkit.h” para que de esta forma sea posible acceder a las diversas funciones del módulo Epanet. Tal como todo programa en lenguaje C que incluye bibliotecas comunes(stdio.h, stdlib.h), el algoritmo evolutivo PEA-WDND incluye también la declaración de la biblioteca Epanet mediante la inclusión de la biblioteca “toolkit.h”

4. Utilizar las funciones de la biblioteca Epanet que se necesiten. Existe información detallada sobre la gran cantidad de rutinas disponibles en Epanet [Rossman, 2000]. El algoritmo evolutivo PEA-WDND utiliza rutinas de Epanet en la función propia *LeeRDA* y *AnalizaRDA*, véase Apéndice II.

En el módulo de optimización desarrollado en este trabajo (Figura 5.3), se encuentra una función llamada *AnalizaRDA*, que es precisamente un intermediario entre el módulo de optimización y el modulo hidráulico, véase 5.4. En esta función se observa que el módulo de optimización realiza llamadas a algunas rutinas del módulo hidráulico. Estas llamadas le permiten al módulo de optimización obtener información sobre la estructura de la red y sobre el comportamiento de la misma. Así, el módulo de optimización puede saber a través del módulo hidráulico cuál es el número de tuberías de la red y sus conexiones; también puede obtener información sobre el número de nodos existentes; saber si los nodos son depósitos o son nodos de consumo; y en general el módulo de optimización puede conocer a través del módulo hidráulico toda la información sobre la red de distribución de agua y su comportamiento.

El módulo de optimización define y envía al módulo hidráulico las configuraciones de la red y el modulo hidráulico le permite conocer las presiones que se tienen en los nodos con dicha configuración. Es importante mencionar que Epanet requiere un archivo de entrada que contenga la configuración de la red en un formato estándar. El archivo de entrada debe contener los datos de la red para una instancia de prueba específica. Este archivo es necesario porque contiene información importante, para Epanet, relacionada con las características de los componentes de la red. Por ejemplo, respecto a las tuberías se define la conexión. Es decir, se indica a que nodos de la red está conectada una tubería. Así mismo se definen las longitudes de las tuberías, los diámetros iniciales, entre otras. En lo que se refiere a los nodos, en el archivo de entrada se puede identificar la ubicación física de los usuarios mediante coordenadas (x, y) definidas. También pueden conocerse las demandas de consumo que tienen los usuarios de la red. Adicionalmente, en el archivo de entrada, se definen las características de las fuentes de abastecimiento, de las bombas de potencia y de cada uno de los componentes de la red. En resumen, el archivo de entrada que requiere Epanet contiene la representación detallada de la red a configurar y las características de los componentes de la red. Una de las funciones más importantes del módulo de optimización es determinar los diámetros de las tuberías que se asignarán a la red, formando una configuración que se envía a Epanet para su validación. Es así como Epanet, es parte importante de este trabajo, ya que trabaja en colaboración con el módulo de optimización para encontrar la mejor solución para el problema de Diseño de Redes de Distribución de Agua. Es importante decir que en este

trabajo primero se verifican las restricciones hidráulicas del modelo de satisfacción de restricciones utilizando Epanet y posteriormente se trabaja en encontrar la solución de costo mínimo utilizando el Algoritmo Evolutivo. Así, se observa interacción constante entre el módulo de optimización y el módulo hidráulico.

Capítulo 5. Desarrollo de un Algoritmo Evolutivo

En este capítulo, se describe la etapa de diseño y desarrollo del algoritmo evolutivo, llamado EA-WDN. Se presentan únicamente las versiones finales tanto del algoritmo evolutivo secuencial como del algoritmo evolutivo paralelo, porque son las que han permitido obtener soluciones de calidad para el problema de Diseño de Redes de Distribución de Agua, abordado en esta tesis. Así mismo, se describen los ajustes paramétricos en las versiones finales, las características específicas y los detalles de implementación de los algoritmos. Se presentan también las consideraciones comunes sobre cada uno de los componentes del algoritmo evolutivo, sus parámetros y su funcionamiento en general, tanto en la versión secuencial como en la versión paralela. Adicionalmente, se describen estudios experimentales realizados sobre la diversidad de la población y la exploración en el espacio de búsqueda. Finalmente se concluye con la explicación del algoritmo evolutivo paralelo funcionando en ambiente Grid, el cual es el resultado de un trabajo extenso, realizado en esta tesis doctoral.

5.2. Estudio Experimental del Algoritmo Evolutivo EA-WDN

En el capítulo 2 se han descrito algoritmos evolutivos desarrollados por diferentes investigadores. En este apartado se describen detalladamente los estudios realizados con el algoritmo evolutivo desarrollado en este trabajo. Los estudios experimentales permiten observar el comportamiento del algoritmo cuando, para cada uno de los operadores y funciones del algoritmo, se utilizan valores y técnicas diferentes. Por ejemplo, en la definición de parámetros el algoritmo puede trabajar con diferentes valores para el tamaño de la población, la probabilidad de cruce, la probabilidad de mutación, el operador de mutación, y el número de generaciones del algoritmo. Esta definición de parámetros se realiza en la versión secuencial del algoritmo y gracias al modelo de paralelización implementado, se realiza también en paralelo.

La sintonización paralela es posible gracias al modelo de paralelización definido en este trabajo. Esta, puede considerarse como una novedosa técnica de sintonización dinámica de parámetros, que permite disminuir el tiempo requerido para la realización de las pruebas experimentales. Una vez definidos los valores de los parámetros del algoritmo, se procede a definir la técnica a utilizar, misma que puede variar en cuanto al tipo de cruce, tipo de mutación, tipo de selección y tipo de remplazo. El tipo de cruce puede ser monopunto o multipunto. El tipo de cruce implica definir un punto de corte a partir del cual se combinarán los cromosomas. Este punto de corte puede ser aleatorio (cualquier valor del cromosoma) o fijo (a la mitad del cromosoma, solo para el cruce monopunto); el tipo de mutación puede ser aleatorio, tanto para definir qué gen sufre una mutación como para definir el valor del nuevo gen mutado; la estrategia de selección, que determina qué individuos pueden convertirse en padres, puede ser aleatoria o torneo. Otro aspecto importante del algoritmo evolutivo desarrollado es la evolución de la población, la cual está estrechamente relacionada con la técnica de remplazo que se utilice. Aquí en este punto el algoritmo desarrollado, PEA-WDND, define si se remplaza la población completa o parte de ella. En este proceso es necesario seleccionar a los individuos que mueren para separarlos de los individuos que continúan viviendo en la siguiente generación. Estas técnicas y aspectos de los algoritmos evolutivos han sido estudiados en diferentes trabajos y problemas pero de forma independiente, es decir implementando una sola técnica a la vez en el algoritmo. En este trabajo doctoral, se hace la implementación de varias técnicas para ver cómo se comportan en la solución un mismo problema. Esta definición de parámetros y definición de estrategias ha sido implementada en el algoritmo evolutivo, tanto en la versión secuencial como en la versión paralela.

Los elementos más importantes del algoritmo evolutivo PEA-WDND son: 1) Representación de la solución, 2) Población inicial, 3) Función de evaluación, 4) Operadores genéticos y 5) Valores para los parámetros del algoritmo (tamaño de población, número de generaciones, probabilidad de cruce, probabilidad de mutación, operador de mutación, entre otros).

5.2.1. Representación de una Solución

El componente básico de los algoritmos evolutivos es el individuo. Éste es una entidad muy importante porque representa la solución de un problema. Además, un conjunto de individuos constituyen una población, la cual es también un elemento primordial de los algoritmos evolutivos. Antes de continuar el estudio de las poblaciones y su evolución, es necesario conocer cuáles son los atributos de un individuo. Existen características propias de cada individuo que permiten definirlo como factible, dependiendo de su capacidad para solucionar un problema, respetando siempre las restricciones establecidas. Por ejemplo, para el problema abordado de diseño de redes de distribución de agua, un individuo es factible si puede satisfacer, al mismo tiempo, tanto las restricciones definidas en el modelo de programación lineal como las restricciones definidas en el modelo de satisfacción de restricciones de la formulación matemática definida en el capítulo 3. Si un individuo como no es capaz de resolver el problema respetando todas las restricciones de ambos modelos entonces se dice que éste es infactible. Finalmente, otro atributo importante cuando se habla de individuos en el algoritmo evolutivo PEA-WDND, es la calidad. Sin embargo, la calidad de un individuo es un concepto relativo en el sentido que éste puede ser “bueno” o “malo”. Una métrica para poder definir la calidad del individuo es definir una función de aptitud con base en la función objetivo del problema, en este caso los individuos buenos serán aquellos que presenten los costos más bajos, recordando que la función objetivo del problema es minimizar el costo del diseño de una red de distribución de agua. De igual manera, los individuos malos serán aquellos que presenten los costos más altos del diseño de la red.

Un individuo o solución para el problema de diseño de redes de distribución de agua, se forma asignando un valor a cada uno de los genes que constituyen el cromosoma, ya sea de forma aleatoria o de forma determinista. Cada individuo tiene, de forma implícita, una aptitud y características que definen su identidad y su calidad. Así, una solución para el problema abordado, se obtiene mediante la asignación de un diámetro de tubería, tomado de la lista de diámetros comerciales disponibles, a cada una de las tuberías de la red, de tal forma que la red resultante cumpla las restricciones de los modelos de satisfacción de restricciones y del modelo de programación lineal definidos

previamente en el capítulo 3. Cada tubería de la red cuenta con un identificador que permite saber a qué nodos de la red se encuentra conectada. Así mismo, cada diámetro de tubería está asociado directamente con un elemento del vector de costos, que ayuda a definir la calidad del individuo, con base en el costo total de la red.

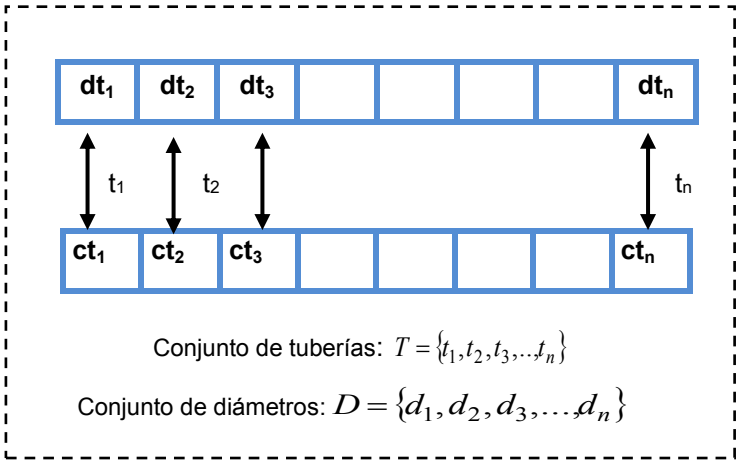


Figura 5.1 Representación de un Individuo

La Figura 5.1 muestra de forma gráfica la estructura de un individuo que será utilizado por los algoritmos secuencial y paralelo, desarrollados en este trabajo. La codificación de una solución, se interpreta como el cromosoma de un individuo compuesto por *n* número de genes, dónde n=8 para este caso específico.

5.2.2. Creación de una Población

Una vez que se definen los atributos importantes del individuo es posible ahora definir a la población como un conjunto de individuos. La creación de una población inicial implica definir diferentes factores y tener presentes algunas consideraciones.

En la creación de una población surgen nuevamente una serie de preguntas [Cruz, 2014]. Por ejemplo, ¿cuál debe ser el tamaño de la población para tener una muestra representativa del problema? Es decir, cuál debe ser el tamaño de la población con la que se trabajará de tal forma que dicha población sea una muestra representativa de la población total, mejor conocida como espacio de búsqueda.

El tamaño del espacio de búsqueda, para el problema de Diseño de Redes de Distribución de Agua, está definido en función del número de tuberías de la red y del número de diámetros comerciales disponibles que pueden ser asignados a las tuberías. Por ejemplo, para las redes de Alperovits, Hanoi y Balerma el espacio de búsqueda es de 14^{8}, 6^{34} y 10^{454} respectivamente.

Otro de los factores a considerar en la creación de la población inicial, es ¿Cómo debe estar constituida una población? Es posible que haya individuos factibles e infactibles, buenos y malos. Sin embargo, estos conceptos son relativos. En este trabajo los conceptos de factibilidad y aptitud se definen en base a las restricciones del modelo matemático y en base a la función de aptitud respectivamente. Aquí lo interesante es saber si la evolución de la población para llegar a la solución óptima depende de la población inicial.

Otra interrogante se refiere a la forma en la que se crea una población inicial. Las formas más comunes de crear una población son la creación de forma aleatoria y la creación determinista, las cuales generan poblaciones con características diferentes y varían también en el tiempo que emplean para la creación. En la población inicial es interesante saber ¿Qué relación existe entre los individuos de la población? Familiares, vecinos, o individuos que provienen de espacios geográficos alejados, en el espacio de búsqueda, y se integran para formar parte de la población; puede existir parentesco entre ellos o no pero cada individuo tiene atributos que lo hacen único y diferente de otros individuos en la población.

Otro punto importante es definir cómo crecerá la población y cómo evolucionará a través del tiempo siguiendo algún método de crecimiento como aritmético, geométrico, entre otros.

Finalmente es importante definir cuáles individuos tendrán descendientes, cuáles individuos sufrirán alguna mutación, y cuáles individuos dejaran de existir.

Las interrogantes descritas en este apartado se han estudiado experimentalmente, mediante técnicas computacionales y han permitido generar un novedoso algoritmo evolutivo que encuentra soluciones de calidad en un tiempo relativamente corto para resolver el problema de Diseño de Redes de Distribución de Agua.

5.2.2.1. Diversidad de una Población

La relación que existe entre los individuos, se maneja en este trabajo como diversidad de la población. Se considera la existencia de individuos diferentes tomados de distintas zonas del espacio de búsqueda para encontrar soluciones de calidad. Se refiere a que los individuos que se convertirán en padres tengan el menor parentesco posible para mejorar la calidad de los descendientes, ya que de acuerdo con los estudios de la genética, se cree que entre menos relación tengan los individuos que serán padres, la descendencia será mejor. Incluso, en este trabajo se ha realizado un estudio experimental en el que se combinan diferentes tipos de individuos y se ha comprobado que existen más probabilidades de generar descendientes factibles cuando los progenitores son factibles y son tomados de zonas alejadas del espacio de soluciones.

5.2.2.2. Generación de Descendientes

La generación de descendientes que constituyen nuevos individuos puede realizarse principalmente mediante la reproducción, al realizar la combinación de dos individuos para generar uno o más descendientes. Sin embargo, en este trabajo se considera también la posibilidad de que una mutación “genere” un nuevo individuo con nuevas características, que al igual que los descendientes tiene la posibilidad de ser elegido para formar parte de una nueva generación. Es importante mencionar que los descendientes se generan mediante la combinación de cromosomas pertenecientes a padres elegidos aleatoriamente. Esto significa que cualquier individuo de la población puede convertirse en padre, incluso aquéllos que tienen las aptitudes más bajas de la población. Es importante mencionar que el algoritmo evolutivo trabaja con individuos factibles. El procedimiento de combinación de individuos se encuentra definido con detalle en 5.3.

5.2.2.3 Remplazo de la Población

La estrategia de remplazo consiste en seleccionar a los individuos que formarán a la nueva población. Esta estrategia determina cuáles serán los individuos que podrán pasar a una siguiente generación. En este trabajo, se han implementado el reemplazo total y el reemplazo parcial, que como su nombre lo indica consisten en remplazar la población completa o parte de ella respectivamente. También se utiliza una técnica de remplazo

elitista que consiste en tomar a los mejores descendientes y nuevos individuos generados mediante los operadores de cruzamiento y mutación.

Los resultados experimentales utilizando diferentes técnicas de reemplazo se presentan en el capítulo 6.

5.2.3. Operadores del Algoritmo Evolutivo

Los operadores del algoritmo evolutivo desarrollado que se utilizan, tanto en la versión secuencial como en la versión paralela, son los operadores de cruce y mutación. El método de combinación de soluciones se realiza mediante el operador de cruce, obteniendo nuevos individuos (hijos o descendientes) mediante la combinación de dos individuos de la población actual (individuos padres).

Por su parte, el operador de mutación "genera" también un individuo nuevo mediante la alteración de la estructura del individuo original. Tanto el operador de cruce como el operador de mutación trabajan mediante probabilidades definidas, con base en el mejor comportamiento del algoritmo, deducido a partir de los resultados experimentales realizados en este trabajo. Las probabilidades se expresan mediante valores que indican el porcentaje de la población que generará descendientes. En la literatura, se han definido para la probabilidad de cruce valores que van desde 50% a 90% de la población. En cuanto a la mutación, generalmente la probabilidad se define como el restante para completar el 100% entre cruce y mutación. Por ejemplo si la probabilidad de cruce se define con 80% entonces la probabilidad de mutación tomará el valor de 20% [Cruz, 2009c]. Este valor indica que de 100 individuos 20 de ellos sufrirán alguna mutación, en el mejor de los casos. En otros trabajos los autores aplican mutación a todos los individuos de la población, generados mediante el operador de cruce [Alba, 2005].

En este trabajo, se ha determinado que un porcentaje de los individuos de la población sufren mutaciones tal como ocurre con la teoría de la selección natural de los seres vivos; incluso la aleatoriedad es la que define cuáles individuos sufrirán alguna mutación y no forzosamente se muta a los descendientes generados mediante el cruce, tal como lo hacen otros algoritmos.

El número de mutaciones que sufre cada individuo puede ser diferente en cada individuo y está definido por el operador de mutación que indica precisamente la probabilidad de que cada uno de los genes, de un cromosoma, sea cambiado. Este operador, generalmente toma valores bajos de 0.03 a 0.05 [Reca, 2008] que representan en un individuo la posibilidad de cambiar únicamente un porcentaje relativamente bajo de los genes. Por ejemplo si el operador de mutación toma el valor de 3% significa que en un cromosoma de 100 genes a lo mucho se cambiarán 3 de esos genes, en el mejor de los casos. En este trabajo se han definido valores desde 0.03 hasta 0.5 para la realización de los estudios experimentales.

5.2.4. Función de Evaluación del Algoritmo Evolutivo

La función de Evaluación del Algoritmo Evolutivo está directamente relacionada con la función objetivo del problema abordado. Recordando que la función objetivo del problema abordado en este trabajo de tesis es minimizar el costo del Diseño de Redes de Distribución de Agua, la función de evaluación determina que los individuos etiquetados como mejores individuos serán aquéllos que presenten las mejores aptitudes. Es decir, para el problema abordado la función de evaluación determina que las mejores soluciones serán aquellas cuyos costos sean los menores. En problemas de maximizar la función de evaluación y la función de aptitud son iguales, pero en problemas de minimizar, como es el caso abordado, se dice que la función de evaluación es la inversa de la función de aptitud, ya que los individuos evaluados con mayor costo corresponden a los individuos menos aptos para ser soluciones del problema.

5.2.5. Parámetros del Algoritmo Evolutivo

El funcionamiento del algoritmo evolutivo depende en gran medida de lograr un balance entre la exploración y la explotación. Este balance se logra con base en la elección correcta de los parámetros de control, principalmente la probabilidad de cruce, la probabilidad de mutación, el operador de mutación y el tamaño de la población, ya que existen interacciones entre ellos [Amor, 2008].

El operador de cruce determina el grado de combinación de los individuos generando nuevas soluciones para el problema. El operador de mutación permite introducción de nuevo material genético en los individuos. Estos dos operadores, fundamentales en los algoritmos evolutivos, contribuyen en gran medida a lograr la evolución de la población.

El tamaño de la población permite encontrar en un grupo de individuos las soluciones más adecuadas para el problema a resolver pero, incrementar el tamaño de una población no necesariamente garantiza que haya diversidad en los individuos de la misma. No obstante, entre mayor es el tamaño de la población mayor es el tiempo que el algoritmo evolutivo secuencial requiere para converger hacia zonas óptimas del espacio de búsqueda. Otro parámetro que debe considerarse es el número de generaciones del algoritmo. Este parámetro es precisamente el que indica el momento en que el algoritmo debe concluir. Es decir, el número de generaciones define la condición de paro del algoritmo y de acuerdo con las pruebas experimentales realizadas en este trabajo de tesis, se ha observado que el tiempo de ejecución del algoritmo evolutivo crece de manera exponencial al incrementar el tamaño de la población y el número de generaciones, véase 6.2.

La selección de los parámetros del algoritmo evolutivo no es una tarea trivial. De hecho, obtener los parámetros adecuados del algoritmo es en sí un problema de programación no lineal [Diego, 2006], que depende de la naturaleza del problema. En la literatura, existen enfoques diferentes para obtener el valor de los parámetros del algoritmo. Por ejemplo, De Jong [De Jong, 1990] utiliza tamaños de población de 100 individuos y probabilidades de cruce y mutación de 0.6 y 0.001 respectivamente. Con estos parámetros, la población conserva en gran medida sus características debido a la escasa actividad de los operadores genéticos. Es decir, con los parámetros definidos la población evoluciona lentamente.

El enfoque propuesto por [Grefenstette, 1986], emplea tamaños de población de 30 individuos y los operadores de cruce y mutación con probabilidades de 0.9 y 0.01. El resultado de establecer estos parámetros es una evolución de la población más rápida gracias a la actividad intensa de los operadores de cruce y mutación. En este trabajo de tesis se han realizado estudios experimentales para encontrar los parámetros que permiten al algoritmo tener un mejor desempeño. Los estudios para definir los valores adecuados para el algoritmo se conocen como análisis de sensibilidad. En este análisis se

han realizado diversos estudios experimentales para encontrar los valores adecuados para el algoritmo evolutivo para diferentes instancias de prueba clasificadas como instancias pequeñas, instancias medianas e instancias grandes, definidas previamente en 3.2. El análisis de sensibilidad se ha realizado en el algoritmo secuencial y al mismo tiempo se consigue un análisis de sensibilidad dinámico como resultado de implementar el modelo maestro-alumnos en el algoritmo PEA-WDND.

5.3. Algoritmo Evolutivo Secuencial EA-WDND

El modelo computacional, Figura 5.2, desarrollado en este trabajo para abordar el problema de optimización de Diseño de Redes de Distribución de Agua, es llamado EA-WDNDM (por las siglas en inglés, Evolutionary Algorithm for Water Distribution Network Model) y está integrado por tres componentes principales, llamados módulos: módulo de optimización, módulo hidráulico y módulo de análisis de resultados. El módulo de optimización consiste de un algoritmo evolutivo, para optimizar el costo del diseño de la red; el módulo hidráulico contiene al simulador Epanet, descrito en el apartado 4.7, para la evaluación de la red; el módulo de análisis de resultados contiene a una herramienta propia, llamada EA-WDNDAnalizer, diseñada para el análisis de los resultados del algoritmo.

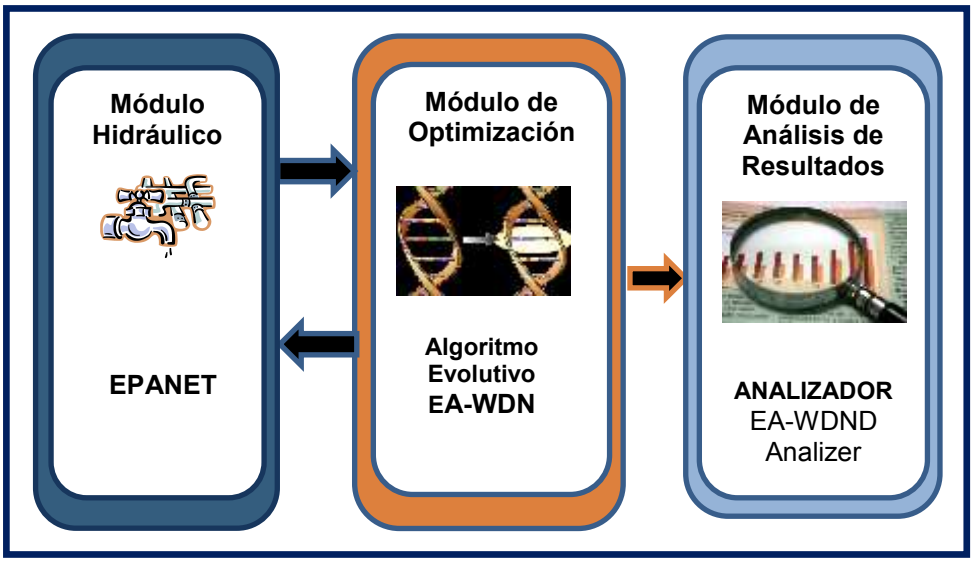


Figura 5.2 Modelo EA-WDNDM

El módulo EA-WDND representa al algoritmo evolutivo desarrollado en lenguaje C y trabajando en ambientes Linux de 64 bits, tanto de manera secuencial como de manera paralela. El módulo EA-WDND puede determinar el diseño óptimo de una red de distribución de agua, trabajando de manera conjunta con el módulo Epanet, el cual es una biblioteca de funciones implantada como una biblioteca estática y enlazada con el módulo de optimización, que interactúa constantemente con el modulo EA-WDND para resolver el problema abordado en esta tesis. Así, el Algoritmo Evolutivo desarrollado es capaz de encontrar soluciones de alta calidad para diseñar o rediseñar redes de diferentes topologías y tamaños.

El analizador EA-WDNDAnalizer es una herramienta propia, codificada mediante un procedimiento almacenado en SQL utilizando, para la codificación y ejecución, la herramienta SQL Server 9.0.5 de Microsoft [SQL, 2005]. Este analizador se encarga del análisis semiautomático de los resultados de los experimentos realizados, generando reportes con los mejores resultados.

Los algoritmos, secuencial y paralelo, ambos desarrollados en lenguaje de programación C, tienen la misma estructura y funcionamiento, con la diferencia que el algoritmo evolutivo paralelo, descrito en el punto 5.4, utiliza la biblioteca de paso de mensajes de MPI, que permite la cooperación de procesos, mediante el envío y recepción de mensajes. El esquema general del algoritmo evolutivo secuencial, desarrollado en este trabajo, puede verse en la Figura 5.3.

El algoritmo evolutivo EA-WDND trabaja con poblaciones de individuos factibles con aptitudes altas. Es decir, la población inicial está formada por los individuos factibles, tomados de una muestra inicial de mayor tamaño que la población, que presenten los costos más bajos para la configuración de la red analizada. Así mismo, la nueva población que se crea en cada una de las generaciones del algoritmo contiene también a los mejores individuos, provenientes de las operaciones de cruce y mutación. La decisión de trabajar con individuos factibles está fundamentada en dos hechos. El primer hecho, es la observación de la evolución natural de los seres vivos, la cual generalmente trabaja con individuos factibles para la generación de descendientes. Es decir, la naturaleza generalmente presenta descendientes de individuos factibles, que se reproducen. Incluso, son casi nulas las probabilidades de que individuos con alteraciones genéticas (llamados infactibles en Algoritmos Evolutivos) pueden reproducirse y dejar descendientes. El segundo hecho está fundamentado en los resultados experimentales realizados en este

trabajo con individuos infactibles, los cuales determinan una lenta y complicada evolución de la población para aproximarse a soluciones factibles y peor aun cuando se busca que la solución factible encontrada sea la solución óptima.

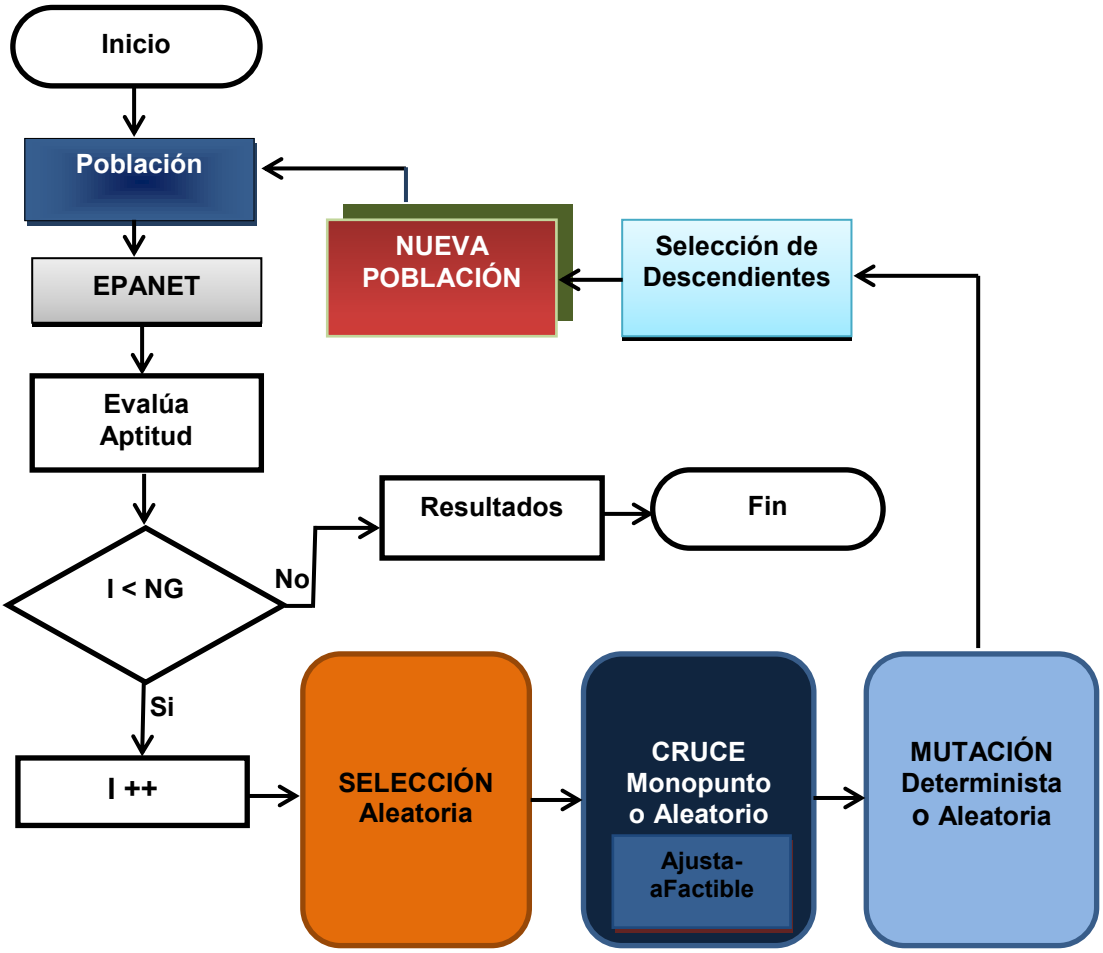


Figura 5.3 Esquema General del EA-WDND

La Figura 5.3 muestra una población inicial que al pasar por el módulo Epanet y el módulo de *EvaluaAptitud* es depurada de tal forma que los mejores individuos o sobrevivientes (*Selección de Descendientes*) forman una población (*Nueva Población*) que incluye únicamente a los individuos factibles, y de ellos a los que presenten mayor aptitud. Con la población factible se aplican los operadores genéticos de selección, cruce y mutación. Al aplicar estos operadores en la población factible se obtiene una nueva población, pero ahora esta población incluye únicamente a los descendientes, nuevamente éstos son individuos factibles con las mejores aptitudes.

El algoritmo repite la evolución de la población hasta alcanzar el criterio de paro definido en la variable *NG*, la cual se refiere al número de generaciones.

.

La representación de un individuo para el problema de Diseño de Redes de Distribución de Agua, para la instancia de prueba Alperovits puede verse en la Figura 5.4. En esta figura, se observan tres vectores: el vector posicionado en la parte superior, que corresponde a las longitudes de las tuberías (en metros) asignadas a la configuración de la red determinadas de antemano por el diseñador de la red; el vector posicionado en la parte central, que corresponde a los diámetros de tuberías asignados a la configuración, tomadas de una lista de diámetros comerciales disponibles (representadas en el conjunto D y asignadas aleatoriamente); el vector posicionado en la parte inferior, que corresponde a los costos comerciales correspondientes a los diámetros de tuberías asignados (representadas en el conjunto C).

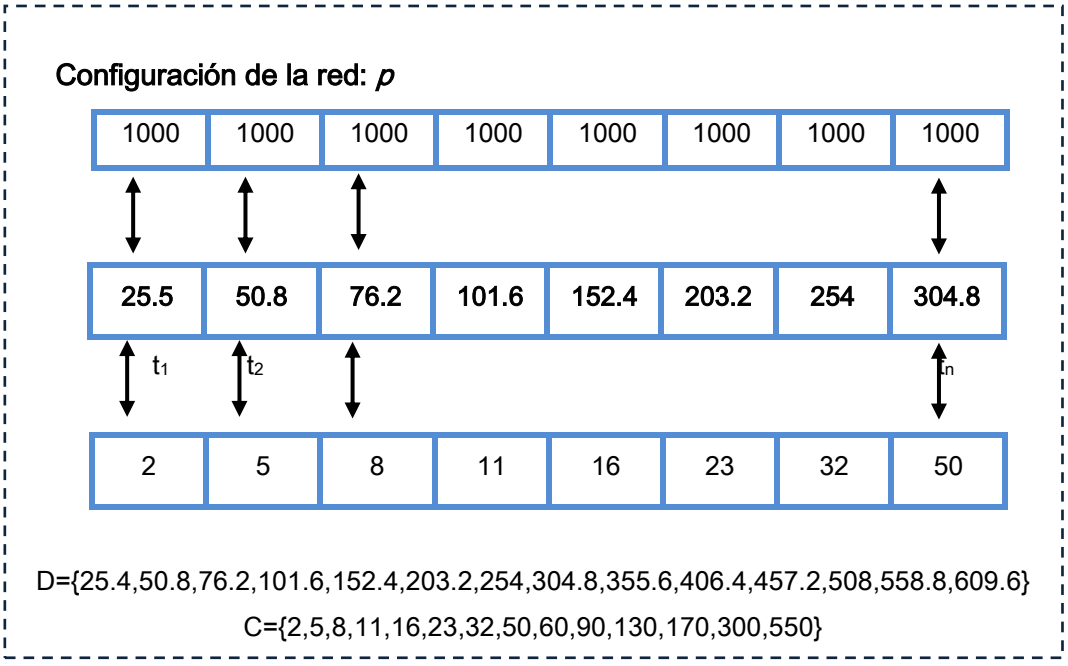


Figura 5.4 Representación de un Individuo para instancia Alperovits

Puede observarse en la Figura 5.4, que para cada elemento del conjunto de tuberías existe un elemento asociado en el conjunto de costos y que los vectores, central e inferior, contienen información de los conjuntos D y C respectivamente.

En resumen, puede decirse que un individuo está formado por un vector de diámetros de tuberías. Cada tubería es asociada con una longitud y un costo, formando así una configuración de la red. De forma implícita, cada configuración de la red tiene un costo total y características hidráulicas que permiten definir si dicha configuración es una solución factible del problema en cuestión, que puede ser implementada en la práctica.

Una vez definida la representación de la solución, el primer procedimiento que se realiza en el algoritmo evolutivo es la creación de la población inicial, la cual se crea generalmente de forma aleatoria, pero en este trabajo la forma de creación de la población se realiza de diferentes formas dependiendo del tamaño de la instancia a resolver. La población inicial ha sido creada de diferentes formas. Incluso para una misma instancia de prueba, como por ejemplo Alperovits, se hicieron estudios experimentales para determinar cómo influye la población inicial en el funcionamiento y convergencia del algoritmo. Así, en este trabajo, algunos estudios realizados para crear la población inicial de forma aleatoria permitieron experimentar el funcionamiento del algoritmo, en sus primeras versiones, trabajando con poblaciones de individuos factibles e infactibles. No obstante, al trabajar con individuos infactibles se observó que el desempeño del algoritmo era deficiente, debido principalmente a que el operador de cruzamiento cuando trabaja con individuos infactibles produce únicamente en promedio 5% de descendientes factibles. Así mismo, se observó que cuando el cruzamiento se realiza con poblaciones de individuos factibles, el 95% de los descendientes generados son factibles. Derivado de estos estudios, el algoritmo evolutivo EA-WDND, desarrollado en este trabajo de tesis, trabaja únicamente con poblaciones de individuos factibles, recordando que los individuos factibles son aquellos que cumplen las restricciones de ambos modelos matemáticos descritos en el capítulo 3.

Los individuos creados en la población inicial deben enviarse al módulo Epanet para determinar si cumplen restricciones hidráulicas y pueden considerarse factibles. Una vez que se determina la factibilidad de los individuos, en el módulo hidráulico, se procede a evaluar cada uno de ellos, mediante una función de evaluación para conocer la aptitud de cada uno de ellos.

En instancias pequeñas, como lo es Alperovits, la creación aleatoria de la población inicial permite tener individuos factibles tomados de diferentes espacios de soluciones, de manera relativamente fácil. Esto se debe a que en la instancia Alperovits, el número de

variables de decisión es pequeño, únicamente 8 tuberías y además de que existen 14 diámetros comerciales que pueden elegirse para la red por lo que la probabilidad de repetir un diámetro de los más pequeños es de 7%. En la Figura 5.5 se muestran los resultados promedio de 30 ejecuciones del algoritmo. Tras realizar las pruebas para generar poblaciones iniciales aleatorias para el espacio de búsqueda de la instancia Alperovits se observó que en una población de 15000 individuos generados de forma aleatoria en promedio, 144 individuos de ellos son factibles. Así, para instancias pequeñas de tamaño similar a Alperovits, es posible generar poblaciones de individuos factibles en tiempos de cómputo limitados. De estas pruebas experimentales realizadas surge la siguiente hipótesis: el espacio de soluciones factibles para el problema de redes de distribución de agua es del 0.01 % del espacio de búsqueda y el tiempo necesario para generar todas las soluciones del espacio de búsqueda utilizando alguna técnica de enumeración sería de aproximadamente 17 días (cada solución toma 0.001 segundos). Esta hipótesis surge mediante un cálculo basado en los resultados de creación de la población mostrados en la Figura 5.5, en la cual se muestra el tiempo requerido para crear una población. Como puede observarse en los resultados, se requiere un tiempo considerable para generar todas las soluciones factibles que se encuentren en el espacio de búsqueda, aun cuando la instancia de prueba es relativamente pequeña, y el problema mayor sería el espacio necesario para almacenar la gran cantidad de información resultante y su análisis posterior. Así se concluye que para generar poblaciones factibles en instancias de prueba pequeñas, la aleatoriedad es una buena alternativa. Sin embargo, en la instancia Hanoi la generación aleatoria para la población inicial arroja resultados poco alentadores, ya que de una muestra de 1,000,000 de individuos generados aleatoriamente, ninguno de ellos es factible. El problema es que a diferencia de la instancia Alperovits, en esta instancia es mayor el número de tuberías que deben configurarse, además de que sólo existen 6 diámetros disponibles comercialmente y con ello la probabilidad de elegir cada diámetro es 16%. Así la configuración de la red de forma aleatoria presenta los diámetros de menor tamaño de la lista de diámetros comerciales presentando redes de costo mínimo pero que no cumplen algunas de las restricciones hidráulicas. Dado que el algoritmo trabaja con poblaciones iniciales factibles, la creación de la población inicial para la instancia Hanoi y Balerma se realiza de forma determinista, asignando a cada una de las tuberías de la red el mayor diámetro disponible comercialmente, ver tabla 2 del apéndice I. Con este procedimiento se obtiene una

configuración que cumple el modelo de satisfacción de restricciones y posteriormente el algoritmo evolutivo se enfoca en lograr el objetivo de minimizar el costo del diseño la red. Antes de empezar a trabajar con la población, se aplica sobre ésta una búsqueda local iterada que permite tener diversidad en la población inicial.

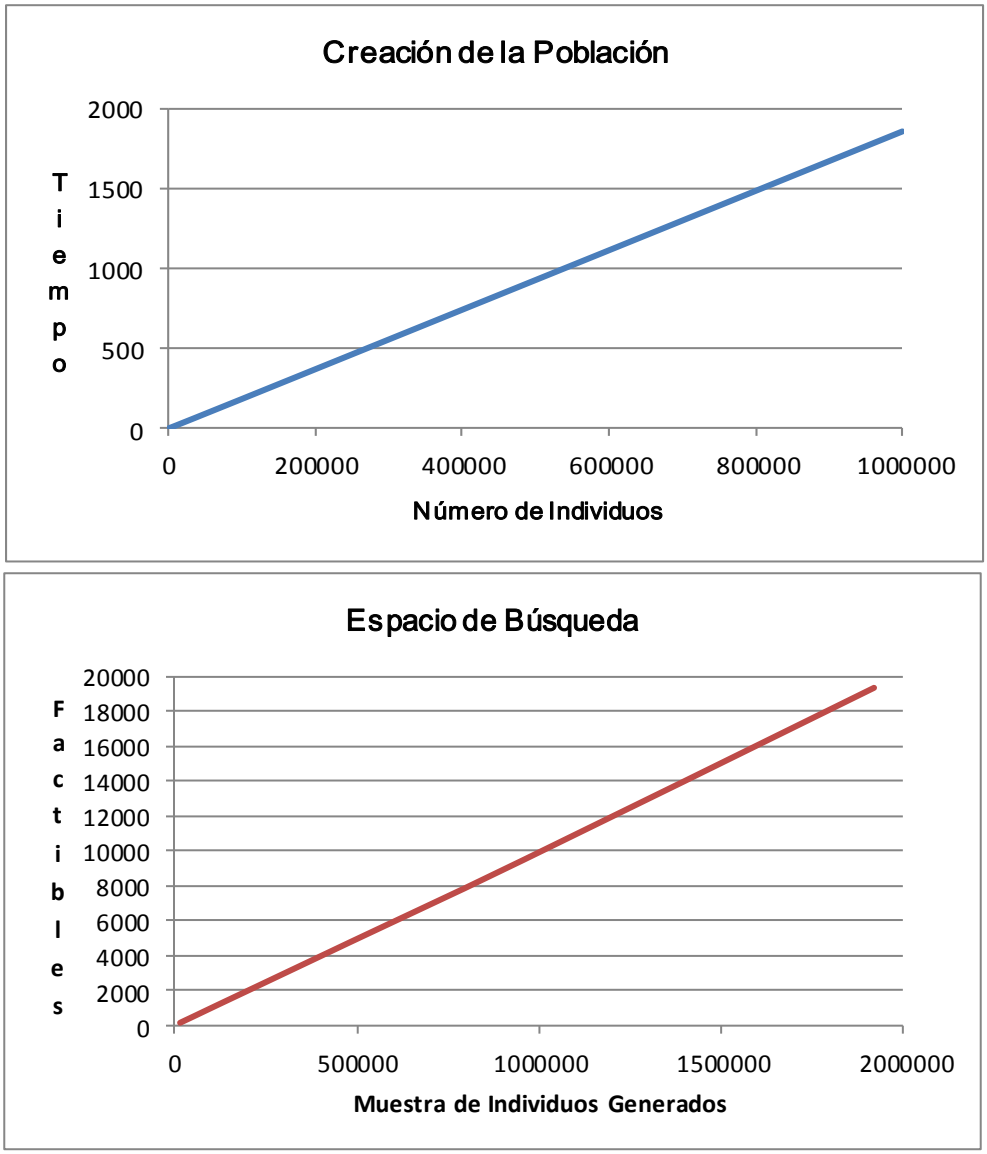


Figura 5.5 Población Factible de la instancia Alperovits

5.3.1. Características generales del Algoritmo Evolutivo EA-WDND

El método de solución desarrollado en este trabajo de tesis, se basa en el esquema general de los algoritmos evolutivos tradicionales. Sin embargo, derivado de esta investigación, se han introducido una serie de mejoras en los componentes y operadores del Algoritmo Evolutivo. Estas mejoras han permitido al algoritmo evolutivo PEA-WDND convertirse en un método de búsqueda eficiente que obtiene soluciones de calidad y en un tiempo de cómputo limitado, para el problema abordado en esta tesis. Los algoritmos secuencial y paralelo pueden ser comparables porque contienen las mismas funciones y la misma estructura. En esta sección se explican las consideraciones comunes del algoritmo evolutivo desarrollado, tanto en versión secuencial como en versión paralela, con la finalidad de no repetirlas en cada sección. En general los algoritmos evolutivos, secuencial y paralelo, desarrollados en este trabajo tienen estructura y características comunes: codificación de las soluciones, creación de la población Inicial, evaluación de restricciones hidráulicas en las soluciones para determinar si éstas son, o no son, factibles, operadores genéticos de selección, cruzamiento y mutación, método para remplazo de individuos y evolución de la población. Así, puede decirse que se trata de un mismo algoritmo definido con las siglas EA-WDND, ejecutándose de forma secuencial y de forma paralela. Sin embargo, se han utilizado las siglas SEA-WDND para hacer referencia al algoritmo evolutivo secuencial y PEA-WDND para referirse al algoritmo evolutivo paralelo funcionando en ambiente Grid.

Los algoritmos SEA-WDND y PEA-WDND son ejecutados en arquitecturas de 64 bits, utilizando características de procesamiento similares, definidas por la infraestructura disponible, véase 6.1. La diferencia importante entre los dos algoritmos es que en el algoritmo PEA-WDND existe la cooperación entre procesos mediante el envío y recepción de mensajes. La estructura general del algoritmo evolutivo EA-WDND puede verse en el algoritmo 5.1. El algoritmo 5.1, muestra un esquema general del algoritmo evolutivo. Este algoritmo necesita como parámetros de entrada saber el número de tuberías de la red (NumTub) y el número de individuos que tendrá la población (NumIndiv), línea 1. El algoritmo inicia con la generación de la población, la cual es aleatoria (algoritmo 5.2) para instancias pequeñas y determinista para instancias grandes, línea 2.

La generación de la población determinista (algoritmo 5.3) consiste en asignar a cada una de las tuberías de la red el diámetro de mayor tamaño de la lista de diámetros comerciales disponibles. Posteriormente se aplica un método de búsqueda local, que corresponde a la función ReasignaDiametros, explicado en el pseudocódigo del algoritmo 5.4.

Algoritmo 5.1. Pseudocódigo del Algoritmo Evolutivo

```
1   Entrada: NumIndiv, NumTub
2   poblacion← GeneraPoblacion(NumIndiv,NumTub)
3   poblacion← BusquedaLocal(poblacion)
4   mientras (cont < NumGen) hacer
5       I ←0
6       mientras (I < NumIndiv) hacer
7           z ← rand [0,100]
8           Indiv1 ← rand [0, NumIndiv -1]
9           si (z < probCr)
10              Indiv2 ← rand [0, NumIndiv -1]
11              pobDescendientes ←CruzaIndividuos(Indiv1, Indiv2)
12          Sino
13              pobDescendientes ←MutaIndividuo(Indiv1)
14          fin si
15          I ←I+1
16      fin mientras
17      cont ←cont +1
18      OrdenaPobDescendientes(0,(ContDescendientes-1));
19      poblacion ← RemplazaIndividuos()
20  fin mientras
21  Devuelve mejor solución
```

La línea 4 define un proceso iterativo para que el algoritmo realice la evolución de los individuos. En este proceso se generan descendientes. La forma de generar descendientes es mediante la aplicación del operador de cruce a una población de individuos (Algoritmo 5.7), línea 3. Puede observarse que algunos individuos de la población son mutados, línea 13 y otros combinados para generar dos descendientes, línea 11. Con los operadores de cruce y mutación se genera una nueva población de individuos, misma que es ordenada de manera ascendente, de acuerdo con las aptitudes,

utilizando el método *QuickSort* por ser un método muy conocido y eficiente de ordenamiento, línea 18. Cabe mencionar que el ordenamiento de datos se hace sobre una tabla indexada que contiene únicamente las aptitudes de los individuos y un índice de enlace que permite relacionar a cada individuo con su correspondiente aptitud, lo cual ayuda a lograr un ordenamiento eficiente y eficaz de los datos. Posteriormente, de la población ordenada se extrae una muestra que contiene a los mejores individuos, línea 19. El algoritmo repite el mismo proceso de evolución de la población, mediante cruces y mutaciones hasta alcanzar el criterio de paro definido en la línea 4. Una vez que se cumple la condición de paro, el algoritmo devuelve la mejor solución encontrada.

Después de realizar múltiples experimentos de ajuste del algoritmo, se encontró que las mejores soluciones se presentan cuando se trabaja con una nueva población obtenida mediante remplazo total de los padres por los descendientes más aptos, por lo que este método de remplazo ha sido implementado en la versión final del algoritmo.

5.3.1.1. Codificación de las Soluciones

Para el problema de Diseño de Redes de Distribución de Agua el cromosoma se representa mediante un vector de flotantes. Esto permite el uso y manipulación de la configuración de manera sencilla, tanto en entornos centralizados como en entornos distribuidos. Esta representación suele ser útil incluso para abordar problemas de gran tamaño, como es el caso de la red Balerma.

5.3.1.2. Creación de la Población Inicial

Los algoritmos EA-WDND, desarrollados en este trabajo utilizan la creación de la población inicial aleatoria para instancias de tamaño pequeño, tal como la instancia Alperovits. El método de creación de población aleatoria permite obtener configuraciones válidas, obtenidas de forma aleatoria. Es decir, el algoritmo *GeneraPoblacionAleatoria* genera individuos factibles que cumplen las restricciones de los modelos matemáticos definidos en el capítulo 3.

Las líneas 5 y 6 del algoritmo permiten hacer la inicialización aleatoria de diámetros de tuberías disponibles para la red. La asignación de diámetros se realiza mediante un recorrido de todas las tuberías de la red. Para asegurarse de que la asignación de diámetros a las tuberías de la red genera una configuración factible. Las líneas 8 y 9 son

llamadas a funciones que realizan el análisis hidráulico de la red para verificar que las soluciones analizadas cumplan las restricciones del modelo de satisfacción de restricciones definido en el capítulo 3. Finalmente, la línea 10 permite guardar en las estructuras de datos del algoritmo, las configuraciones factibles, mismas que pueden ser soluciones para el problema de diseño de la red.

Algoritmo 5.2. Pseudocódigo de la generación de población aleatoria

```
1   Entrada: NumIndiv, NumTub
2   NumFact ← 0
3   mientras (NumFact < NumIndiv) hacer
4       I ← 0
5       mientras (I < NumTub) hacer
6           Config[I] ← rand [0,NumDiam-1]
7       fin mientras
8       si LeeRed (NombreInstancia) entonces
9           si AnalizaRed(Config) entonces
10              poblacion←GuardaSolucion(Config)
11              NumFact ← NumFac +1
12          fin si
13      fin si
14  fin mientras
```

La generación de la población ha sido definida para *n* individuos factibles representados con la variable NumIndiv, siendo NumIndiv un número entero finito. NumIndiv puede tomar valores de 2 a 10,000. Para definir el tamaño de población ideal, para que el algoritmo tenga un mejor desempeño, se hizo un estudio de análisis de sensibilidad de los parámetros del algoritmo definiendo diferentes tamaños de población, véase 6.2.1. Es importante mencionar que para instancias como Alperovits se crea una población de mayor tamaño, por ejemplo 1000, 10000 individuos y de ella se extrae una muestra de los mejores individuos para formar la población inicial de tamaño definido mediante el análisis de sensibilidad.

Otra forma de generar la población inicial en los algoritmos cuando se diseñan redes de distribución de agua de gran tamaño (red Balerma que cuenta con 454 tuberías), es mediante un procedimiento híbrido que consiste en la asignación de diámetros de tuberías

determinista combinado con una búsqueda local iterada, descrito en el algoritmo 5.3. A diferencia de la creación de la población inicial aleatoria, el procedimiento para generar la población factible se realiza en dos etapas. En la primera etapa se inicializa la configuración de la red con el diámetro mayor disponible (líneas 3-4) asegurando con ello que la solución sea factible en el modelo de satisfacción de restricciones, según los requisitos del problema. En una segunda etapa, se hace una re-asignación, que consiste en realizar de forma aleatoria cambios a ciertas tuberías de la red. El número de tuberías a cambiar está definido mediante la variable *NumCambios*. Una parte fundamental del funcionamiento correcto del algoritmo es validar que las configuraciones generadas cumplan las restricciones hidráulicas y con ello puedan considerarse soluciones factibles (líneas 8-13). El algoritmo repite el procedimiento para generar cada individuo, hasta completar el tamaño de la población definida previamente, mediante la variable *NumIndiv*.

Algoritmo 5.3. Pseudocódigo de la generación de población determinista

```
1  Entrada: n, NumTub
2  I, NumFact ← 0
3  mientras (I < NumTub) hacer
4          Config[I] ← Diametros[NumDiam-1]
5  fin mientras
6  mientras (NumFact < n) hacer
7          ConFiguracion ← ReasignaDiametros(Config,NumCambios,)
8          si LeeRed (NombreInstancia) entonces
9                  si AnalizaRed(Config) entonces
10                         GuardaSolucion(Config)
11                         NumFact ← NumFac +1
12                 fin si
13         fin si
14 fin mientras
```

El procedimiento que reasigna diámetros para generar diversidad en la población consiste en generar cambios de forma aleatoria pero clasificando ahora a los diámetros en tres conjuntos. Es decir el vector de diámetros es dividido de forma implícita en tres partes para contener así a los conjuntos de diámetros “pequeños”, “medianos” y “grandes”. De tal forma que al reasignar un diámetro de tubería (línea 5) se genera un evento de forma aleatoria se pueda elegir alguno de los conjuntos previamente

clasificados y posteriormente, es un segundo evento el que determina qué elemento del conjunto se debe seleccionar. Una vez que se obtiene una nueva configuración, ésta se envía al analizador Epanet para verificar si es o no factible.

La creación de forma aleatoria tiene como ventaja que genera configuraciones válidas en un tiempo relativamente corto, por lo que puede utilizarse para instancias pequeñas como es el caso de Alperovits. No obstante, para instancias de tamaño grande, como Balerma, es más conveniente crear la población inicial determinista ya que, a diferencia de la creación aleatoria, permite generar configuraciones factibles en tiempo de cómputo limitado. Las líneas 8-9 de los algoritmos 5.2 y 5.3 requieren de la evaluación de la configuración en el módulo hidráulico por lo que se hace uso de las funciones *LeeRed* y *AnalizaRed*. Este proceso se repite cierto número de veces, para mejorar la solución, de acuerdo con la variable *NI,* la cual indica precisamente el número de intentos que debe realizar la función ReasignaDiametros, algoritmo 5.4, para obtener una mejor solución que la que recibe como parámetro de entrada. Al final del proceso el algoritmo devuelve como salida una solución mejorada.

Algoritmo 5.4. Pseudocódigo ReasignaDiametros

```
1   Entrada: solución, NumCambios,NI
2   config ← solución
3   mientras (I < NI) hacer
4       mientras (NumCambios != Ø) hacer
5           config← reasignar_diametro_tuberia
6       fin mientras
7       factible← AnalizaRDA(config)
8       I ← I+1
9       si (factible y config_es_ mejor_ que_ solución )
10          solucion← config
11      si no
12          ReasignaDiametros;
13      fin si
14  fin mientras
15  Devolver solución
16  Salida: una solución mejorada
```

5.3.1.3. Evaluación de las Configuraciones

La evaluación de las configuraciones de la red consiste en verificar que se cumplan las restricciones definidas en el modelo matemático de satisfacción de restricciones, véase 3.3.2. El pseudocódigo descrito en el algoritmo 5.5 permite obtener información de la red que se analizará.

El algoritmo requiere que se especifique como parámetro de entrada el nombre de la instancia a resolver, misma que debe estar guardada en un archivo con extensión ".inp", con un formato específico establecido por la biblioteca Epanet (línea 2).

El módulo Epanet genera archivos de reporte y de salida, en los cuales se muestran los resultados del comportamiento de la red. Como puede observarse, la mayoría de las instrucciones utilizadas en el algoritmo *LeeRed* contienen el prefijo *EN*, el cual que indica que es una función propia de la biblioteca Epanet. El algoritmo 5.5 permite conocer el número de tuberías y el número de nodos de la red a configurar (líneas 8 y 9). Epanet obtiene esta información mediante un análisis del archivo de entrada .inp.

Algoritmo 5.5. Pseudocódigo LeeRed

```
1   Entrada: ArchivoEnt
2   CodigoError ← ENopen(ArchivoEnt, ArchivoRep, ArchivoSal)
3   si (CodigoError >0) entonces
4           ENclose()
5           return 0
6   sino
7           ENopenH()
8           ENgetcount(EN_LINKCOUNT,NumTub)
9           ENgetcount(EN_NODECOUNT, NumNodos)
10  fin si
11  return1
```

Una vez obtenidas las características de la red, Epanet procede con el análisis de la configuración indicada para verificar que ésta cumpla las restricciones hidráulicas, mediante el pseudocódigo *AnalizaRed* definido en el algoritmo 5.6. Este algoritmo, permite trabajar de manera conjunta con el módulo de optimización para determinar cuál es la mejor solución para el diseño de una red de distribución de agua. El algoritmo *AnalizaRed* verifica que la configuración dada cumpla las restricciones definidas en el

modelo satisfacción de restricciones mientras que el módulo de optimización, definido en el algoritmo 5.1, se encarga de asegurar que la configuración cumpla las restricciones del modelo de programación lineal.

El algoritmo *AnalizaRed* requiere como parámetro de entrada una configuración con diámetros que se asignarán a cada una de las tuberías de la red.

El primer paso del algoritmo *AnalizaRed* es realizar un recorrido en la red para simular la asignación física de un diámetro a cada una de las tuberías a la red (líneas 4-6). Una vez que se asigna el diámetro a las tuberías de la red, Epanet verifica que las presiones en los nodos de consumo de la red sean superiores a las presiones mínimas requeridas definidas en la variable *PresionMinReq*, líneas 9-22.

Algoritmo 5.6. Pseudocódigo AnalizaRed

```
1   Entrada: Configuracion,NumTub, NumNodos
2   I, Cumple ← 1
3   NumDebMin, J,K← 0,
4   mientras (I < NumTub) hacer
5           ENsetlinkvalue(I+1, EN_DIAMETER,Configuracion[I])
6   fin mientras
7   ENinitH(0)
8   ENrunH(t)
9   mientras (J < NumNodos) hacer
10          presión,K ← 0
11          ENgetnodetype(J, tipoNodo)
12          si (tipoNodo == 0) entonces
13                  ENgetnodevalue(J, EN_PRESSURE, presion)
14                  si ( presion < PresionMinReq ) entonces
15                          NumDebMin← NumDebMin+1
16                          Cumple← 0
17                          Presiones[K] ← presion
18                          K ← K+1
19                  fin si
20          fin si
21          J ← J+1
22  fin mientras
23  ENclose()
24  ENcloseH()
25  devolver Cumple
```

Un aspecto importante en el análisis de la red es saber si un nodo de la red es de consumo o es un depósito (línea 11). Una vez que se sabe que un nodo es de consumo Epanet verifica (línea 14) que en dicho nodo existan las presiones mínimas requeridas comparando estas presiones con un valor de referencia previamente establecido para la red analizada, véase Apéndice I. El proceso de análisis de una red se repite para todas las configuraciones que el módulo de optimización defina, cuidando siempre que todas las configuraciones sean válidas en el modelo de programación lineal. La función *AnalizaRed* en la salida, línea 25, indica si una configuración cumple con las restricciones del modelo de satisfacción de restricciones, pero también indica mediante la variable *NumDebMin* el número de nodos de la configuración que no alcanzan las presiones mínimas requeridas, lo cual es importante para que una solución infactible se ajuste a factible en algún momento posterior.

5.3.1.4. Operadores Genéticos

Los operadores genéticos utilizados en el algoritmo evolutivo EA-WDND, son comunes tanto en la versión secuencial como en la versión paralela. En esta sección se explican las características de los operadores que utiliza el algoritmo evolutivo, iniciando la explicación con el operador de selección.

La selección de los individuos que serán combinados se realiza de forma aleatoria, aunque también se han definido la selección por torneo, la selección elitista y la selección por ruleta. Sin embargo, se ha observado que la selección aleatoria es la que ofrece los mejores resultados para la convergencia del algoritmo. Ya que tal como se menciona en la literatura, este tipo de selección permite que cualquier individuo pueda generar descendientes, lo cual ayuda a mantener la diversidad de la población.

En el caso de la selección por ruleta, y elitista se ha observado que los individuos más aptos son los que tienen mayores probabilidades de generar descendientes, ya que son favorecidos en la selección. La gran desventaja es que los mejores individuos se reproducen con mayor frecuencia que los individuos menos aptos generando mayor número de descendientes que heredan sus características. Con esto la población

contiene individuos que predominan ocasionando que la diversidad de la población sea cada vez menor. En este trabajo, la selección aleatoria se apega a la idea de que cualquier individuo puede ser elegido para su reproducción.

La combinación de individuos para generar descendientes se hace mediante el operador de cruce. En concreto, este trabajo de tesis utiliza el cruce monopunto y multipunto con tipo de corte aleatorio. El número de combinación de individuos está directamente relacionado con la probabilidad de cruce p. Algunos autores aplican cruces a una población de tamaño n, lo cual significa que puede haber p combinaciones en el mejor de los casos, normalmente con valores de 60% a 90%.

En otros trabajos el número de combinación de individuos está definido por el tamaño de la población, ya que se hacen múltiples combinaciones hasta que el número de descendientes sea igual al tamaño de la población con la que trabaja el algoritmo. En este trabajo, se hacen múltiples combinaciones para la población de acuerdo con los valores de la probabilidad de cruce, véase 6.2. Con ello, el número de descendientes, generados por las combinaciones de individuos, es mayor que el tamaño de la población con la que trabaja el algoritmo, teniendo así la oportunidad de elegir de esa nueva población a los mejores descendientes. Incluso se hicieron pruebas en las que, en una generación, se aplicaban recursivamente los operadores de cruce y mutación a los individuos, con probabilidades de cruce y mutación definidas. Esto significa que algunos individuos se combinaban varias veces generando mayor cantidad de descendientes. Como resultado de este experimento se obtenía una población dos o tres veces mayor que con la población con la que se trabaja. De la población resultante se tiene mayor número de individuos para elegir solo a los mejores. Con este experimento, se observó que efectivamente se mejoraba la calidad de las soluciones, apartado 6.3.

El operador de cruce consiste en lo siguiente: se seleccionan dos individuos I_1 e I_2 de la población, de longitud L, que actuarán como progenitores; se escoge de manera aleatoria un escalar $k \in [1,L)$. Este valor es el denominado punto de cruce y sirve para indicar la división del cromosoma para generar dos soluciones “hijas” formadas de la siguiente manera: la primera poseerá los k primeros genes de I_1 y el resto del I_2, la otra solución hija se formará de manera inversa: con los k primeros genes de I_2 y los restantes de I_1, algoritmo 5.7.

Algoritmo 5.7. Pseudocódigo del Cruzamiento

```
1   Entrada: Config1, Config2
2   I, Cumple← 1
3   NumDebMin, J,K← 0,
4   mientras (I < PuntoCruce) hacer
5           Individuo1[I] ← Config1[I])
6           Individuo2[I] ← Config2[I])
7           CostoIndiv1[I] ← CostoConfig1[I]
8           CostoIndiv2[I] ← CostoConfig2[I]
9   fin mientras
10  I ← PuntoCruce
11  mientras (I < NumTuberias) hacer
12          Individuo1[I] ← Config2[I])
13          Individuo2[I] ← Config1[I])
14          CostoIndiv1[I] ← CostoConfig2[I]
15          CostoIndiv2[I] ← CostoConfig1[I]
16  fin mientras
17  NumDebMin← AnalizaRDA(Individuo1)
18  si (NumDebMin ==0)
19          GuardaDescendiente(Individuo1, CostoIndiv1)
20  sino
21          si NumDebMin <NumTuberias/10
22                  si AjustaaFactible
23                          GuardaDescendiente(Individuo1, CostoIndiv1)
24          fin si
25  NumDebMin← AnalizaRDA(Individuo2)
26  si (NumDebMin ==0)
27          GuardaDescendiente(Individuo2, CostoIndiv2)
28  sino
29          si NumDebMin <NumTuberias/10
30                  si AjustaaFactible
31                          GuardaDescendiente(Individuo2, CostoIndiv2)
32          fin si
33  Salida: dos descendientes
```

Otro de los operadores comunes, en los algoritmos desarrollados, es la mutación. La mutación es una operación básica en los algoritmos evolutivos, porque ayuda a realizar una explotación del espacio de búsqueda, permitiendo mejorar los individuos de una población. En este trabajo se han realizado pruebas con el operador de mutación,

variando el método de mutación. Es decir, se han realizado pruebas experimentales utilizando la mutación determinista para el problema abordado en la tesis, que consistía en aumentar o disminuir un diámetro de tubería [Baños, 2010b]. No obstante, la experiencia ha demostrado que el método de mutación que permite obtener resultados es la mutación aleatoria. Por tanto, se ha definido mutación determinista y aleatoria en los algoritmos desarrollados en este trabajo de tesis, tal como lo describe el algoritmo 5.8.

Algoritmo 5.8. Pseudocódigo de la Mutación

```
1   Entrada: Indiv, NumTub, ProbMuta
2   mientras (I < NumTub) hacer
3       aleatorio ← [0,1]
4       si (aleatorio< ProbMuta) entonces
5           tipoMuta ← [1,3]
6           si (tipoMuta == 1) entonces
7            Tuberia ← AumentaDiametro
8           fin si
9           si (tipoMuta == 2) entonces
10           Tuberia ← DisminuyeDiametro
11          fin si
12          si (tipoMuta == 3) entonces
13           NuevoDiametro ← rand [0,NumDiametros]
14           Tuberia ← NuevoDiametro
15          fin si
16          I← I+1
17      fin si
18  fin mientras
19  Salida: un nuevo individuo
```

El algoritmo de mutación recorre los elementos de la configuración, línea 2, para decidir de acuerdo con la probabilidad de mutación si un gen se muta o no. En caso de mutarse se tienen tres tipos de mutación. El tipo de mutación se elige de forma aleatoria y puede consistir en aumentar diámetro (líneas 6-8), disminuir diámetro (líneas 9-11) o cambiarlo por otro diámetro definido de forma aleatoria (líneas 12-15).

El proceso de mutación genera individuos nuevos, con características y aptitud definidas. Los individuos mutados son clasificados en nuevo grupo, junto con los descendientes resultantes en la operación de cruce. El siguiente paso del algoritmo evolutivo es el remplazo de individuos que consiste en actualizar la población con los nuevos integrantes. Generalmente, los algoritmos evolutivos tradicionales utilizan técnicas de remplazo que consisten en que los descendientes remplazan a los padres solo si mejoran su aptitud. En este trabajo, para la actualización de la población se utiliza una técnica, que no fue encontrada en la literatura, por lo que es una propuesta en este trabajo de tesis.

La técnica mencionada consiste en formar un grupo de todos los descendientes generados gracias a los operadores de cruce y mutación, donde ambos se aplican a diferentes individuos de la población. Este grupo, generalmente es de mayor tamaño que la población con la que trabaja en algoritmo, ya que en cada generación se realizan *n* operaciones de cruces o de mutaciones, donde *n* es el tamaño de la población. Así al tener una población (formada por los descendientes y los nuevos individuos mutados) de mayor tamaño que la población con la que trabaja el algoritmo, se genera competencia entre los individuos, por lo que los más aptos logran sobrevivir una generación más. Es decir, los descendientes más aptos son seleccionados para formar una nueva población que vivirá en la siguiente generación del algoritmo. Cabe mencionar que los padres mueren y son solo los hijos quienes viven una nueva generación, en la cual pueden ser combinados para generar descendientes o bien sufrir alguna mutación que los permite convertirse en nuevos individuos, o incluso pueden mantenerse sin cambio alguno. Esta técnica de remplazo simula la forma de reproducción de los insectos, en donde una generación pone huevos, se aleja geográficamente o muere y es substituida por una nueva.

Es importante decir que para mantener fijo el tamaño de la población, generación tras generación, utilizando la técnica de remplazo total es necesario que el número de individuos factibles generados mediante el cruce y la mutación sea mayor que el tamaño de la población. En caso de que esta condición no se cumpla solo podría hacerse un remplazo parcial. Con el remplazo parcial de la población coexistirán padres e hijos y

existiría la posibilidad de que se combinen para generar descendientes, lo cual no ocurre en la naturaleza ni en este algoritmo desarrollado.

Esta técnica de remplazo surge de la observación de que cuando se remplazaban a los padres por el descendiente más apto después de su nacimiento se tenía el problema de que en una población había individuos repetidos que no ayudaban a continuar con la evolución natural del algoritmo, porque había réplicas exactas del mismo individuo.

Uno de los problemas más comunes que presentan los algoritmos evolutivos es la pérdida de información. Es decir, cuando se hace la combinación de individuos para generar descendientes o bien cuando se hacen mutaciones sobre ellos, en repetidas ocasiones los nuevos individuos no son factibles, y éstos se desechan sin importar el grado de infactibilidad que presenten. Por ejemplo, un individuo que representa la configuración de una red puede violar las restricciones de presiones en la red en uno o en dos nodos, con ello se considera que el individuo es infactible y se desecha, cuando en la mayoría de los nodos la red de distribución de agua el servicio es adecuado para los usuarios de la red. Cuando el costo computacional de generar nuevos individuos es segundos, como lo es el caso de la red Alperovits, desechar individuos infactibles no representa gran problema, ya que se pueden generar nuevos individuos con relativa facilidad. No obstante, cuando se trata de redes de gran tamaño como lo es el caso de Balerma, desechar soluciones y generar nuevas soluciones afecta el desempeño del algoritmo, ya que generar individuos factibles representa una tarea costosa. En este punto se ha considerado enfocarse en esta debilidad de los algoritmos evolutivos, de desperdiciar información que puede ser importante, y ver la forma de que los individuos infactibles se conviertan en factibles. Para lograr este objetivo se ha desarrollado el algoritmo 5.9. Este algoritmo se encarga de identificar cuáles nodos de la red de distribución de agua no ofrecen servicio adecuado a los usuarios de la red, porque no respetan las presiones mínimas requeridas. Así, la función *AjustaaFactible*, descrita en el pseudocódigo del algoritmo 5.9, se encarga de analizar las configuraciones y redimensionar la red de tal forma que todos los usuarios de la misma tengan al menos las presiones mínimas requeridas que garantizan un servicio y funcionamiento adecuado de toda la red.

El algoritmo 5.9 permite cambiar algunas tuberías de la red por otras, con diámetros diferentes, con la finalidad de que todos los usuarios de la red tengan finalmente un servicio adecuado.

Algoritmo 5.9. Pseudocódigo AjustaaFactible

```
1   Entrada: Config, CostoConfig, CostoTotal,NumTub, NumNodos ,NumDebMin
2   si ajustes == 0
3           ConfigNuevaRed ← Config
4           CostosConfigNuevaRed ← CostoConfig
5           CostosNuevaRed ← CostoTotal
6   fin si
7   CostoNuevaRed←CostoConfigR
8   ConfigTmp←Config
9   CostosConfigmp← CostoConfig
10  I, Cumple ← 1
11   J,K← 0,
12  mientras (I < NumTub) hacer
13          si presionesConfig[I] <PresionMinReq
14                  posNodo ←BuscaConexionesNodo(I)
15                  TuberiaAnterior ←(random(TubIncidentes(posNodo))
16                  TuberiaNueva ←CambiaDiametro(TuberiaAnterior)
17                  ConfigTmp← ConfigTmp(TuberiaNueva)
18                  si AjustaaFactible(ConfigTmp)
19                          NumDebMin←0
20                          ConfigNuevaR← ConfigTmp
21                          AjustaFact← 1
22                  si no
23                          AjustaaFactible(ConfigNuevaR)
24                  fin si
25          fin si
26  fin mientras
27  Devuelve AjustaFact
28  Salida: un valor boleano que indica si la red funciona correctamente
```

El primer paso para poder redimensionar una red es identificar el número de nodos de la red que tienen un funcionamiento inadecuado, con presiones inferiores a las mínimas requeridas por los usuarios. Esta información se puede conocer mediante la variable *NumDebMin*.

Posteriormente, es necesario identificar cuáles son los nodos de la red que presentan presiones inferiores a las presiones mínimas requeridas por los usuarios de la red (línea 13).

A continuación es necesario identificar cuáles son las tuberías que dan servicio al usuario para cambiar, al menos, una de ellas por otra de tamaño distinto al tamaño actual (líneas 14-15). Con este cambio se obtiene una nueva configuración con una tubería distinta (línea 17). Esta nueva configuración es ahora analizada por el simulador Epanet para verificar si el usuario tiene un servicio adecuado y también verificar que con el cambio no haya aumentado el número de nodos de la red con presiones inferiores a las mínimas requeridas. Si el cambio de diámetro en la tubería no mejora el funcionamiento de la red, se realiza el cambio nuevamente hasta lograr que la red funcione correctamente.

La función ajusta a factible permite dar mantenimiento a redes de distribución de agua existentes cambiando en ella algunas tuberías por otras tuberías con diámetros distintos de tal forma que la nueva red pueda brindar a los todos los usuarios un servicio acorde a sus necesidades. Con este algoritmo no solo se ajusta el servicio de una red, sino que también puede utilizarse para el tema de rediseño de Redes de Distribución de Agua que estén operando de forma incorrecta o que estén dando un mal servicio en la distribución de agua a los usuarios, lo cual es también una aportación de este trabajo de tesis. Esta aportación es una ventaja en el algoritmo PEA-WDND porque permite corregir el problema que generalmente presentan los algoritmos evolutivos tradicionales, lo cual se refiere a la tendencia de desechar las soluciones infactibles resultantes de cruces y mutaciones sin importar el grado de infactibilidad que presenten, presentando mayor costo computacional y afectando la eficacia del algoritmo.

5.4. Algoritmo Evolutivo Paralelo PEA-WDND

En este apartado, se presenta la estructura y las características del algoritmo evolutivo paralelo desarrollado en este trabajo, para abordar el problema de Diseño de Redes de Distribución de Agua. El algoritmo evolutivo desarrollado en esta tesis doctoral, gracias a su naturaleza, se adapta con relativa facilidad al modelo de paralelismo utilizado en este trabajo.

Los algoritmos evolutivos secuencial y paralelo presentan estructura y características comunes, con la diferencia de que en este último existe cooperación entre los procesos mediante el envío y recepción de resultados, con base en la implementación de un modelo híbrido que surge de la idea de combinar el modelo maestro-esclavos con el modelo de islas. Así, el algoritmo evolutivo paralelo fue diseñado e implementado mediante el modelo maestro-alumnos el cual presenta las bondades de ambos modelos, para formar un nuevo modelo novedoso (resultado de múltiples estudios y experimentos) que presenta excelentes resultados para el problema de Diseño de Redes de Distribución de Agua.

El Algoritmo Evolutivo paralelo fue desarrollado en el lenguaje de programación C y utilizando la biblioteca de paso de mensajes MPI compatible con las versiones de MPI de Intel [MPI, 2013], OpenMPI [OpenMPI, 2015] y MPICH [MPICH, 2013] que implementan el estándar MPI-1 [MPI, 2013] y MPI-2 [MPI2, 2013]. Así mismo el algoritmo utiliza la biblioteca de funciones de Epanet.

El modelo resultante, propio de este trabajo, es llamado modelo maestro-alumnos ya que realmente es una analogía del funcionamiento que ocurre en grupos, de maestros y alumnos, en el ámbito escolar. Al igual que en el modelo maestro-esclavos, en el modelo maestro-alumnos se tiene un proceso maestro para controlar la ejecución del programa asignando trabajo a los alumnos, pero en lugar de tener un solo maestro, se tienen tantos maestros como grupos se definan, así cada grupo se tiene un maestro Figura 5.6.

Sea $GR=\{Gr_1,Gr_2,Gr_3,...,Gr_{NumGr}\}$ el número de grupos que se definen para la ejecución del algoritmo, donde NumGr es el número de grupos. El conjunto GR se representa por Gr={A,p∈P}. El conjunto está constituido por $A=\{a_1,a_2,a_3,...,a_{nA}\}$ donde cada elemento del conjunto A, conocido como alumno, equivale a un esclavo del modelo maestro-esclavos y nA es el número de alumnos que tendrá cada grupo. El conjunto Gr es una función biyectiva con el conjunto $P=\{p_1,p_2,p_3,...,p_{nP}\}$, $f:Gr\rightarrow P$. Es importante decir que cada conjunto A ∈ Gr es distinto en cada conjunto Gr. El tamaño para el conjunto Gr, está definido mediante TamGr=|GR|, y del conjunto A, NumAlum=|A|.

Los tamaños de los conjuntos deben definirse, antes de la ejecución del algoritmo evolutivo en ambiente Grid.

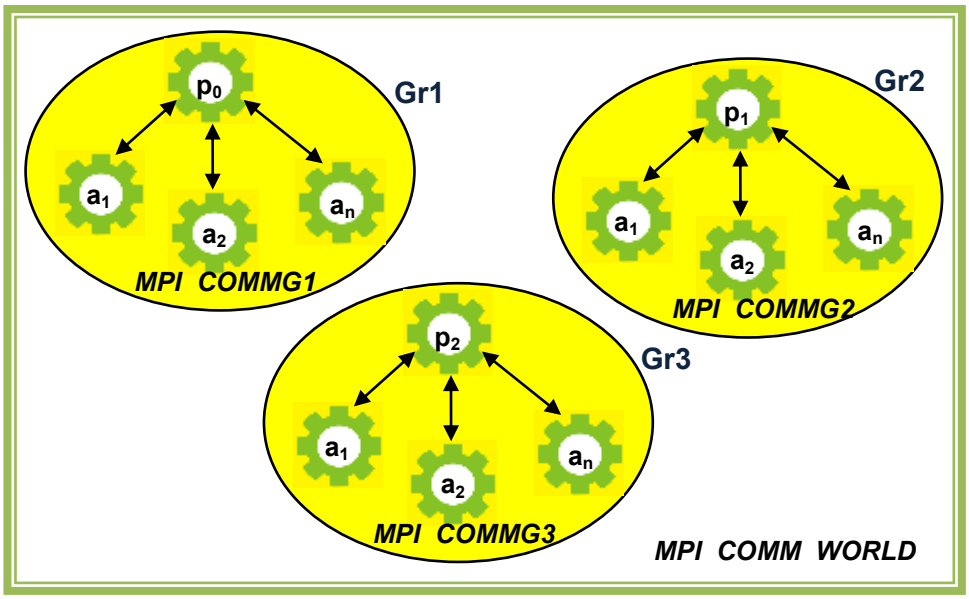


Figura 5.6 Modelo Maestro-Alumnos

Una vez definidos los valores de los parámetros del algoritmo evolutivo paralelo (tamaño de los conjuntos), inicia la ejecución del algoritmo con la implementación del modelo maestro-alumnos. En este modelo, el proceso maestro realiza un conjunto de acciones y los procesos alumnos pueden realizar acciones idénticas pero sobre datos diferentes. Este modelo, se ha implementado de tal manera que los alumnos definen su propia estrategia de solución para el problema asignado. De esta forma, es posible tener tantas estrategias diferentes de solución como número de alumnos haya, ya que es poco probable que dos alumnos resuelvan el problema utilizando exactamente la misma estrategia de solución. Cabe mencionar que una estrategia de solución propuesta por un alumno se refiere a un algoritmo evolutivo pero con diferentes parámetros para tamaño de población, probabilidad de cruce, probabilidad de mutación, entre otros. La estrategia también puede incluir valores o técnicas diferentes en lo que respecta a la selección, tipo de cruce, mutación, remplazo de la población, entre otras, Figura 5.7.

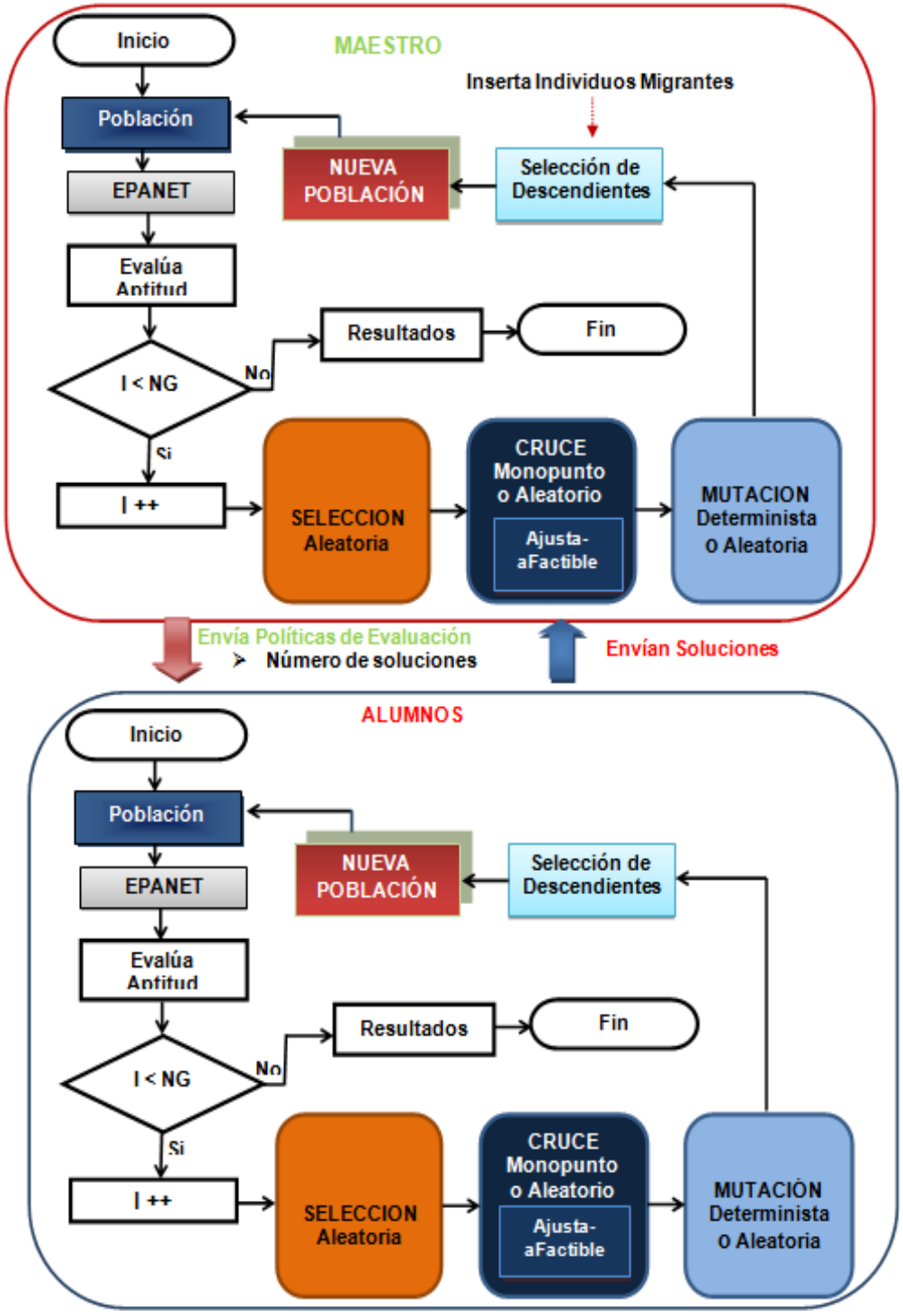


Figura 5.7 Diagrama del Algoritmo Evolutivo Paralelo

Aparentemente las actividades que realizan los alumnos y maestros parecen iguales. Sin embargo, éstos tienen funciones definidas y diferenciadas. Por ejemplo, el maestro asigna tareas a los alumnos, define parámetros que indican la forma de trabajo. Entre otros, define las reglas de tiempos de entregas, los formatos de envío de soluciones. El maestro da a conocer a los alumnos los valores definidos en dichos parámetros. El nodo maestro resuelve también el problema para poder verificar las soluciones de los alumnos. El maestro recibe las soluciones de los alumnos en un formato codificado que contiene a la solución junto con su costo, en un vector de flotantes. Finalmente, el maestro decodifica las soluciones y las inserta en su población de descendientes para poder después realizar algún tipo de remplazo (total o parcial) y trabajar con una nueva población compuesta principalmente por las soluciones de los alumnos.

Los alumnos reciben el problema a resolver de los maestros, y las reglas de tiempos y formatos de entrega, definen su estrategia de solución, resuelven el problema y entregan resultados.

Se dice que el maestro es el responsable de la inicialización, la coordinación, la sincronización y las operaciones de salida mientras que los alumnos son responsables de resolver el problema asignado, y dar a conocer las soluciones al maestro, mediante el uso de algoritmos evolutivos.

Al igual que el modelo maestro-esclavos, el modelo implementado en este trabajo es un paradigma fácil de visualizar desde la perspectiva de administración de algoritmos y también es relativamente fácil de implementar. Este modelo permite mantener la secuencia del algoritmo original pero la administración del algoritmo es realizada por cada maestro de un grupo, dividiendo la carga de trabajo en subgrupos basándose en el modelo de islas. El modelo de islas se basa en la idea intuitiva de dividir una población en subpoblaciones para distribuirlas entre un conjunto de procesos disponibles. Sin embargo, en el modelo implementado en este trabajo cada proceso crea su propia población. En el caso de los alumnos ocasionalmente emigran individuos hacia el proceso maestro, con cierta frecuencia. Así, del modelo de islas se toma la idea de las comunicaciones para implementarla en el modelo maestro-alumnos con las siguientes políticas:

1) **Número de emigrantes** (m), se define un porcentaje de la población como límite máximo Num_{sol} de soluciones a enviar, de tal forma $m \in \{0,1,2,3,..., Num_{sol}\}$ .Así, el valor que se asigna a m es generado de forma aleatoria tomando un valor del

conjunto. Los estudios experimentales con este parámetro definieron a Num_{sol} con 1% del tamaño de la población.

2) **Frecuencia de migración** (r), $r \in \{0,1,2,3,..,NumGen\}$: es el número de generaciones realizadas en una isla sin comunicarse con otras para enviar individuos. Así el parámetro r o T_e define la frecuencia de migración, la cual se determina de forma aleatoria con un valor del conjunto definido. Por ejemplo si el valor de r=10 y el número de generaciones del algoritmo evolutivo es *NumGen*=100, entonces se enviarán soluciones al maestro cada 10 generaciones, realizando así 10 envíos durante la ejecución del algoritmo.
3) **Envío de migrantes** (S): se refiere a los individuos que se envían de una isla a otra. Para definir cuáles individuos migrarán se debe elegir, de entre varios métodos disponibles en la literatura, un método de selección, por ejemplo selección aleatoria, selección elitista, selección por torneo, entre otras. En este trabajo se utiliza la selección elitista.
4) **Reemplazo de migrantes** (R): esta política define la integración de los migrantes en la nueva subpoblación. En ocasiones, consiste en reemplazar a los peores individuos de una isla por los mejores individuos de otra isla. En este trabajo de tesis se hace un remplazo total o parcial con los mejores descendientes.
5) **Sincronización**. Indica el tipo de envío/recepción, que puede ser bloqueante o no bloqueante, entre las islas. También define en qué momento se deben integrar los migrantes a la nueva población, puede ser en cuanto llegan o en algún momento posterior de la búsqueda. En este trabajo, el remplazo se hace cuando el maestro completa la recepción de tareas de los alumnos.

Tal como ocurre en el modelo de Islas, los alumnos envían diferente número de soluciones (Num_{sol}), cada cierto número de generaciones (T_e), ambos parámetros son definidos por el maestro en las reglas de evaluación. En este modelo, las políticas de comunicación, de envío y de recepción de resultados para la evaluación del trabajo de los alumnos, son definidas por el maestro del grupo y pueden variar en cada grupo obteniendo diferente comportamiento del algoritmo. La idea de establecer las políticas de evaluación en el modelo maestro-alumnos, es con la finalidad de lograr una sincronización natural aprovechando los beneficios de tener una plataforma Grid homogénea, ya que

todos los procesos terminan las actividades en tiempos similares, evitando tiempos de ocio en los procesos.

Una consideración muy importante para plantear la sincronización, radica en el hecho de que el proceso maestro no puede permanecer en espera de respuestas durante minutos. El tiempo definido son a lo sumo segundos, ya que en caso contrario se ocasiona una terminación temprana sin éxito del algoritmo por no recibir respuesta por uno o más nodos.

En el algoritmo evolutivo paralelo implementado bajo el esquema de maestros-alumnos, al definir el tiempo de entrega se soluciona el problema de sincronización de procesos, ya que los alumnos que no terminan en el tiempo establecido simplemente no pueden entregar sus tareas porque el maestro del grupo ya no estará esperando la entrega. Así que los alumnos deben esperar al nuevo tiempo de entrega para entregar los nuevos resultados a tiempo.

En el modelo maestro-alumnos las comunicaciones son esporádicas. Prácticamente, en toda la ejecución del algoritmo se realizan tres comunicaciones entre el maestro y los alumnos. Inicialmente el proceso maestro establece la comunicación con los alumnos para dar a conocer el problema a resolver y las políticas de evaluación. La segunda vez que se comunican, son los alumnos quienes establecen la comunicación con el maestro para notificarle la entrega de resultados. Finalmente, los alumnos establecen nuevamente una comunicación con el maestro para enviarle las soluciones del problema. La comunicación que se establece, entre maestros y alumnos, es punto a punto bloqueante y necesita un tiempo de envío y de recepción, tal como lo muestra la Figura 5.8.

Figura 5.8 Comunicación entre maestro y alumnos

El primer paso para la implementación del modelo maestro-alumnos, consiste en diseñar grupos de procesos que pueden comunicarse entre sí, tal como lo describe el algoritmo 5.10. El algoritmo necesita saber el tamaño de los grupos que se realizarán, lo cual está definido en la variable de entrada TamGrupo. Es necesario inicializar el ambiente de ejecución MPI para poder hacer uso de las rutinas definidas en la biblioteca de funciones de MPI, línea 2. Una vez inicializado el ambiente MPI es importante conocer el identificador de los procesos y el número de procesos disponibles, líneas 3-4.

Algoritmo 5.10. Pseudocódigo del Algoritmo Evolutivo Paralelo

```
1   Entrada: TamGrupo
2   MPI_Init
3   IdProceso←MPI_Comm_rank()
4   NumProcesadores←MPI_Comm_size()
5   NumGrupos←NumProcesadores/ TamGrupo
6   mientras (I < NumGrupos) hacer
7           LimiteInferior ← I*( TamGrupo)
8           LimiteSuperior ← (I+1)*( TamGrupo)
9           si (IdProceso>= LimiteInferior && IdProceso < LimiteSuperior)
10                  Grupo←I+1;
11                  Alumnos ← Alumnos(LimiteInferior, LimiteSuperior)
12          fin si
13  fin mientras
14  NuevoComunicador←Grupo
15  MPI_Comm_group(MPI_COMM_WORLD, &orig_group)
16  MPI_Group_incl(orig_group,NumIndivGrupo,Grupo, &new_group)
17  MPI_Comm_create(MPI_COMM_WORLD, new_group, & NuevoComunicador)
18  IdSubProceso← MPI_Group_rank()
19  si (IdProceso == 0)
20          Maestros←AsignaMaestros(IdProceso)
21          funcion←1
22  fin si
23  si(función == 1)
24          RealizaActividadesMaestro()
25  si no
26          RealizaActividadesAlumno()
27  Salida: grupos definidos y comunicadores para los grupos
```

La creación de los grupos está definida en las líneas 6-13. Aquí se determina cuáles elementos pertenecen al grupo y se obtiene un identificador de grupo, línea 10, que se utilizará para la creación de un comunicador grupal, línea 14.Con este identificador de grupo se obtiene el nuevo comunicador grupal, líneas 15-17, para posteriormente obtener un identificador de proceso grupal, que incluye y permite comunicar a todos los procesos del grupo. Las líneas 19-22 sirven para que el proceso maestro global obtenga una lista de procesos para los diferentes grupos. Cabe mencionar que la selección de maestros la realiza aleatoriamente y la asignación a los alumnos es en base a la cercanía del identificador local de los procesos. Una vez que el maestro global determina a los procesos que serán maestros de grupo, éste da a conocer a los alumnos del grupo su correspondiente profesor, línea 23. Finalmente los alumnos conocen a su profesor y asumen su función de maestro o alumno dentro del grupo, líneas 24-26.

El pseudocódigo del algoritmo 5.10 permite la creación de los grupos de trabajo compuestos por un maestro y uno o más alumnos. También permite indicar a los procesos su función dentro del grupo, para que éstos asuman su rol de maestro o alumno.

El algoritmo evolutivo paralelo PEA-WDND, con la implementación del modelo maestro-alumnos, tiene diferentes poblaciones que evolucionan de manera independiente. Ocasionalmente, se producen migraciones entre las poblaciones para intercambiar material genético. Con la utilización de la migración, este modelo puede explotar las diferencias en las subpoblaciones. Esta variación representa una fuente de diversidad genética. El maestro se encarga de asignar a los alumnos un problema que deben resolver. Así mismo, el maestro define las reglas de evaluación que los alumnos deben respetar. Las comunicaciones pueden repetirse con una frecuencia establecida por el profesor del grupo. Es importante decir que la primera comunicación de maestros a alumnos se establece al inicio de la ejecución del algoritmo. Las comunicaciones posteriores, de alumnos a maestro, están definidas con base en la asignación de políticas que el maestro estableció para la evaluación. Entre otras reglas, el maestro define el número de soluciones que los alumnos le deben entregar para cada problema y el formato que deben tener las soluciones antes de enviarlas. El maestro también establece los tiempos de entrega en que los alumnos deben enviarle el trabajo. Es decir, si el maestro define que los alumnos deben enviarle las soluciones en un tiempo de entrega T_e, entonces en el algoritmo se establece que cada T_e generaciones el alumno envía al

maestro una notificación, en caso de tener resultados que se enviarán al maestro, y seguida de ella iniciará el envío del paquete serializado que contiene las soluciones del problema. El procedimiento de las actividades que realiza el proceso maestro, están definidas en el pseudocódigo del algoritmo 5.11. En la línea 1 y en la línea 3 el maestro determina el número de soluciones que espera recibir y la fecha de entrega, respectivamente. En la línea 8 puede observarse el envío de un vector de tipo entero que el profesor envía a sus alumnos. En este vector es precisamente donde el profesor indica al alumno el número de soluciones que debe enviar y la fecha de entrega de dichas soluciones. En la primera generación del algoritmo, líneas 12-16, se genera una población factible, generalmente de mayor tamaño a la población con la que trabaja el algoritmo. Posteriormente se ordena de manera ascendente con base en la aptitud de los individuos. De ésta, se extrae una muestra que forma la primera población en la cual se aplican los operadores de selección, cruce y mutación para obtener descendientes y nuevos individuos, líneas 17-29. Los nuevos individuos son colocados en un nuevo conjunto de descendientes para posteriormente elegir de ellos a los más aptos para formar una nueva población, mediante una técnica de remplazo, líneas 24, 26 y 37. En generaciones posteriores, la población con la que trabaja el algoritmo se obtiene también mediante una técnica de remplazo, línea 38. En las líneas 30-35 el profesor verifica si ya es tiempo de recibir soluciones de los alumnos, así mismo revisa las notificaciones de los alumnos que enviarán resultados, ya que puede ocurrir que algunos alumnos, por diferentes causas no entreguen resultados. Posteriormente, el profesor se prepara para la recepción de soluciones de los alumnos y almacena los resultados para poder hacer uso de éstos. Una vez finalizada la recepción, el profesor crea una nueva población con soluciones de diferentes alumnos y registra una nueva fecha de entrega de soluciones (línea 34). El proceso se repite hasta alcanzar la condición de paro del algoritmo, definida por el número de generaciones, línea 17. De todo este procedimiento el resultado final es una solución que representa el mejor costo del problema a resolver, que en este trabajo es la solución del problema de Diseño de Redes de Distribución de Agua.

Algoritmo 5.11. Pseudocódigo del Algoritmo Evolutivo Paralelo: RealizaActivMaestro

```
1  cont ← 1,NumSoluciones← GeneraNumAleatorio(0,TamPoblacion/10)
2  poblacion← GeneraPoblacion(n,NumTub)
3  TiempoEntrega←GeneraNumAleatorio(1,NumGeneraciones/3)
4  AgendaTE← TiempoEntrega
5  poblacion← BusquedaLocal(poblacion)
6  para I ←0 hasta NumIndivGrupo
7      si (IdSubProceso != I)
8          MPI_Send(&Actividades,2,MPI_INT,I,etiqueta,NuevoCom)
9      fin si
10 fin para
11 si LeeRDA
12     si cont == 1
13                 GeneraPoblacionFactible()
14                 OrdenaPoblacionFactible()
15                 CopiaSubconjunto()
16     fin si
17     mientras (cont <= NumGen) hacer
18         I ←0
19         mientras (I < n) hacer
20             z ← rand [0,100]
21             Indiv1 ← rand [0,n-1]
22             si (z < probCr)
23                 Indiv2 ← rand [0,n-1]
24                 pobDescendientes ←CruzaIndividuos(Indiv1, Indiv2)
25             Sino
26             pobDescendientes ←MutaIndividuo(Indiv1)
27             fin si
28             I ←I+1
29         fin mientras
30     si (AgendaTE == I)
31         RecibeNotificacion()
32         RecibeResultados()
33         AlmacenaResultados(NumSolucionesRecibir)
34         AgendaTE← AgendaTE+TiempoEntrega
35     fin si
36     cont ←cont +1
37     OrdenaPobDescendientes(0,(ContDescendientes-1));
38     RemplazaIndividuos()
39     fin mientras
40 fin si
```

El pseudocódigo del algoritmo 5.12 muestra el proceso de recepción de resultados que realiza el maestro del grupo. En la línea 2 se define el número de individuos que enviaran soluciones al maestro. En la línea 3 se establece la condición para que el maestro reciba resultados de los diferentes procesos alumnos.

Algoritmo 5.12. Pseudocódigo del Algoritmo Evolutivo Paralelo: RecibeResultados

```
1  Entrada: un vector que contiene una solución
2  para K ←0 hasta NumIndivGrupo
3        si (IdProceso != K)
4        MPI_Recv(&Resultado,NumElem,MPI_FLOAT,K,etiqueta,NuevoCom, &estado)
5        para I ←0 hasta NumSolRecibir
6              para J ←0 hasta NumTuberias
7                    Poblacion[P]. ConfigRed[J]←Resultado[L]
8              fin para
9              para J← NumTuberias hasta (2* NumTuberias)
10                   Poblacion[P]. CostoConfigRed[J]←Resultado[L]
11             fin para
12       Aptitudes[P].CostoTotalConfigRed←Resultado[L]
13       fin si
14       fin para
15 fin para
```

El envío y recepción de resultados se hace a través del vector *Resultado*, el cual es un vector de flotantes, definido globalmente ya que en MPI las variables que se utilicen para el envío y recepción de datos deben estar definidas de manera global para el funcionamiento correcto del programa y para el almacenamiento adecuado de los datos. Esto no se dice en la literatura, pero de acuerdo con la experiencia se ha llegado a esa conclusión. El vector *Resultado* contiene un conjunto de datos codificados de la siguiente manera. Suponiendo que se trata de la instancia Alperovits, donde una solución tiene 8 tuberías, y el maestro estableció que el alumno debe enviarle 5 soluciones cada 4 generaciones del algoritmo (Tiempo de Entrega), entonces el vector *Resultado* tiene longitud, almacenada en la variable *NumElem*, con valor de 85 que será enviada cada cuatro generaciones. Este vector tiene una configuración de 8 tuberías, 8 costos de las tuberías y un costo total de la configuración. Es decir, cada solución para el problema tiene en total 17 posiciones del vector. Con esta codificación se tiene el beneficio de realizar un solo envío a través de la red, con ello se evitan comunicaciones frecuentes que saturan la red y afectan la eficiencia del algoritmo. En instancias pequeñas como

Alperovits, no se aprecia la importancia de la codificación, pero en redes como Balerma, el beneficio es fácilmente apreciado. Otra ventaja de esta codificación es que se envían los costos de las soluciones, evitando que el maestro tenga que hacer cálculos antes de poder utilizar los individuos (soluciones) en una nueva generación para la evolución de la población. Esta representación codificada se estableció como el resultado de una serie de experimentos, de los cuales se observó que se gana eficiencia en el algoritmo.

En el modelo maestro-alumnos, propuesto en este trabajo e implementado con el algoritmo desarrollado PEA-WDND, los alumnos por su parte reciben, de su respectivo maestro, los problemas que deben resolver. Cada alumno, tal como ocurre en el ámbito escolar, define su propia estrategia de solución para resolver el problema. Los alumnos pertenecen a un grupo y tienen un único maestro a quien deben enviar, en el tiempo que él establece, los resultados que obtienen al resolver el problema, algoritmo 5.13.

En la línea 3, del algoritmo 5.13, puede observarse la recepción de un vector de tipo entero que es enviado por el maestro del grupo. En este vector es precisamente donde el maestro indica al alumno el número de soluciones que debe enviar y la fecha de entrega de dichas soluciones. En la primera generación del algoritmo, líneas 8-12, se genera una población factible, generalmente de mayor tamaño a la población con la que trabaja el algoritmo. A continuación, se ordena esta población de manera ascendente con base en la aptitud de los individuos. Posteriormente, se extrae de la población de mayor tamaño una muestra que forma la primera población en la cual se aplican los operadores de selección, cruce y mutación para obtener descendientes y nuevos individuos, líneas 13-25. Los nuevos individuos son colocados en un nuevo conjunto de descendientes para posteriormente elegir de ellos a los más aptos para formar una nueva población, mediante una técnica de remplazo, líneas 20,22, 27 y 28. En generaciones posteriores, la población con la que trabaja el algoritmo se obtiene mediante una técnica de remplazo, línea 28. En las líneas 29 a 33 se verifica si ya es tiempo de entregar soluciones al maestro. En caso de que ya se haya llegado la fecha de entrega, el alumno notifica al maestro el envío de las soluciones. Después, el alumno hace el envío de soluciones (línea 31) y registra una nueva fecha de entrega de soluciones (línea 32). De todo este procedimiento el resultado es el envío de una o más soluciones, según lo especifique el maestro.

Algoritmo 5.13. Pseudocódigo del Algoritmo Evolutivo Paralelo: RealizaActividadesAlumno

```
1   cont ← 1,poblacion← GeneraPoblacion(n,NumTub)
2   poblacion← BusquedaLocal(poblacion)
3   MPI_Recv(&Actividades,2, MPI_INT,profe,etiqueta,NuevoCom, &estado)
4   TiempoEntrega ← Actividades[0]
5   NumSolucionesEnviar← Actividades[1]
6   AgendaTE← TiempoEntrega
7   si LeeRDA
8       si cont ==1
9           GeneraPoblacionFactible()
10          OrdenaPoblacionFactible()
11          CopiaSubconjunto()
12      fin si
13      mientras (cont <= NumGen) hacer
14          I ←0
15          mientras (I < n) hacer
16              z ← rand [0,100]
17              Indiv1 ← rand [0,n-1]
18              si (z < probCr)
19                  Indiv2 ← rand [0,n-1]
20                  pobDescendientes ←CruzaIndividuos(Indiv1, Indiv2)
21              sino
22              pobDescendientes ←MutaIndividuo(Indiv1)
23              fin si
24              I ←I+1
25          fin mientras
26      cont ←cont +1
27      OrdenaPobDescendientes(0,(ContDescendientes-1));
28      RemplazaIndividuos()
29      si (AgendaTE == I)
30          EnviaNotificacion()
31          EnviaResultados(NumSolucionesEnviar)
32          AgendaTE← AgendaTE+TiempoEntrega
33      fin si
34      fin mientras
35  fin si
36  Salida: mejor solución de cada generación
```

En el modelo maestro-alumnos podría ocurrir que los alumnos se comunicaran entre ellos. No obstante, esta es una limitación del algoritmo PEA-WDND.

La ventaja principal de que los alumnos no se comuniquen entre ellos, en el algoritmo PEA-WDND, es la diversidad de las soluciones. En un trabajo futuro podría establecerse comunicación entre alumnos o entre maestros y aumentar la experimentación en esta área. El algoritmo PEA-WDNA establece la comunicación exclusiva en grupos, de alumnos con maestro, con el objetivo principal de que los alumnos le envíen al maestro las soluciones del problema, pseudocódigo 5.14.

Algoritmo 5.14. Pseudocódigo del Algoritmo Evolutivo Paralelo: EnviaResultados

```
1  NumElem← (2*NumTuberias+1)*NumSolEntregar
2  TiempoEntrega←GeneraNumAleatorio(1,NumGeneraciones/3)
3  AgendaTE← TiempoEntrega
4  poblacion← BusquedaLocal(poblacion)
5  para I ←0 hasta NumSolEntregar
6          para J ←0 hasta NumTuberias
7                  Soluciones[k]←solucion[I].ConfigRed[J]
8          fin para
9          para J ← NumTuberias hasta (2* NumTuberias)
10                 Soluciones[k]←solucion[I].CostoConfigRed[L]
11         fin para
12         Soluciones[k]←solucion[I].CostoTotalConfigRed
13 fin para
14 MPI_Send(&Resultado,NumElem, MPI_FLOAT,profe,etiqueta,NuevoCom
```

Como resultado final de la implementación del modelo maestro-alumnos, propuesto en este trabajo, se tiene el beneficio de poder estudiar el comportamiento del algoritmo de forma amplia utilizando diferentes estrategias, propuestas todas ellas por los alumnos. Otra de las ventajas del modelo maestro-alumnos, es la división de trabajos en grupos, de un tamaño que puede definirse. El maestro de cada grupo obtiene los mejores resultados de los alumnos. Una ventaja más es que al tener un maestro por grupo, se evitan las comunicaciones frecuentes hacia un solo maestro y se dividen hacia varios maestros. Otra ventaja notable del modelo, es que la codificación de soluciones para el envío y recepción de resultados permite disminuir el número de comunicaciones ayudando también a mejorar la eficiencia del algoritmo evolutivo PEA-WDND.

Capítulo 6. Análisis de Resultados

En este capítulo, se valida experimentalmente el desempeño del algoritmo evolutivo, desarrollado en este trabajo y descrito en los capítulos precedentes, para encontrar soluciones de calidad para el problema de Diseño de Redes de Distribución de Agua. El capítulo se divide en cuatro secciones. La primera sección describe las especificaciones de hardware y software de la plataforma utilizada en la experimentación. La segunda sección presenta los experimentos previos a la ejecución del algoritmo evolutivo, con la finalidad de tener un ajuste paramétrico que le permita al algoritmo un mejor funcionamiento, tanto en eficiencia como en eficacia para las instancias de prueba resolver. La tercera sección muestra los resultados obtenidos con el algoritmo evolutivo paralelo, descrito en el capítulo previo. Finalmente, en la cuarta sección se establecen dos comparaciones: la primera comparación se realiza entre los algoritmos evolutivos secuencial y paralelo desarrollados en este trabajo de tesis, y para ratificar el cumplimiento de los objetivos planteados al inicio del presente trabajo, la segunda comparación se realiza con los resultados de otros investigadores que abordan el mismo problema. El plan experimental que se presenta en este capítulo incluye: experimentos para obtener los parámetros del algoritmo. Los experimentos incluyen diferentes métodos de tipos de selección, tipos de cruzamiento, tipos de mutación y tipos de reemplazo. Cabe mencionar que los experimentos se realizan con instancias pequeñas, medianas y grandes, mostradas en el apéndice I. Así mismo, los experimentos se realizan con el algoritmo evolutivo paralelo y con el algoritmo evolutivo secuencial, ambos desarrollados, para resolver el problema de Diseño de Redes de Distribución de Agua, en el presente trabajo de tesis doctoral. Es importante mencionar que todos los experimentos se realizan durante 30 ejecuciones, bajo las mismas condiciones, debido al carácter no determinista de los algoritmos evolutivos. En los resultados se presentan los valores máximo, mínimo y promedio de las ejecuciones del algoritmo dando así mayor confiabilidad y sustento al trabajo de tesis realizado.

6.1. Especificaciones de la Plataforma de Experimentación

La plataforma de cómputo utilizada para la experimentación y la ejecución de los algoritmos es la Grid Morelos [Cruz, 2012b]. La Grid consiste en el enlace de los clúster de alto rendimiento, Cuexcomate, y Texcal.

Figura 6.1 Clúster Cuexcomate

Figura 6.2 Clúster Texcal

El clúster cuexcomate, Figura 6.1, se encuentra en el laboratorio de optimización y software dentro del centro de investigación en Ingeniería y Ciencias Aplicadas de la Universidad Autónoma del Estado de Morelos, y el clúster Texcal, Figura 6.2, se encuentra localizado en Jiutepec Morelos, en la Universidad Politécnica del Estado de Morelos. Actualmente la miniGRID cuenta con 58 nodos, con los cuales se tiene un total de 186 núcleos de procesamiento, con una capacidad de 171 GB de RAM, 34 Terabytes de almacenamiento en disco duro. Adicionalmente se tienen 2 nodos GPU, con 1792 núcleos de procesamiento, 72GB de RAM y 2 Terabytes almacenamiento en disco duro. El apéndice III muestra las especificaciones de los clúster Cuexcomate y Texcal respectivamente [Cruz, 2012b].

6.2. Análisis de Sensibilidad del Algoritmo Evolutivo EA-WDND

El análisis de sensibilidad, o problema de configuración de parámetros [Eiben, 1999], consiste en realizar un ajuste paramétrico, para definir los valores de los parámetros del algoritmo evolutivo. Este ajuste es imprescindible a fin de poder observar la eficiencia del algoritmo y establecer los valores de los parámetros que forman la mejor configuración de parámetros para el funcionamiento del algoritmo. El ajuste de parámetros es muy importante para que el algoritmo evolutivo pueda ofrecer su mejor desempeño en la resolución del problema abordado, ya que el éxito del algoritmo evolutivo en la solución de un problema depende en gran medida de una configuración adecuada de los parámetros del algoritmo [Eiben, 2014].

El análisis de sensibilidad del algoritmo EA-WDND consiste en realizar múltiples estudios experimentales asignando en cada estudio diferentes valores a los parámetros del algoritmo. La finalidad es encontrar los valores para cada uno de los parámetros con los cuales el algoritmo es eficiente y eficaz en la solución del problema de Diseño de Redes de Distribución de Agua. El algoritmo evolutivo EA-WDND desarrollado en este trabajo de tesis ha sido el resultado de una gran cantidad de estudios experimentales. Incluso, se puede decir que el algoritmo se ha desarrollado en varias etapas, con base en los resultados de los estudios experimentales. Los experimentos más importantes,

llevados a cabo en esta investigación, se centran en el problema del ajuste paramétrico del algoritmo, el cual consta de dos etapas. En la primera etapa, se analiza el comportamiento del algoritmo evolutivo al variar parámetros básicos tales como el tamaño de la población, la probabilidad de cruce, la probabilidad de mutación y el número de generaciones. En esta etapa, los valores de los parámetros del algoritmo pueden establecerse de manera estática o de manera dinámica, según se requiera. Una vez establecidos los valores de los parámetros del algoritmo en la primera etapa, se realizan estudios experimentales para observar el comportamiento del algoritmo evolutivo variando, en una segunda etapa, el tipo de combinación de los individuos, el tipo de remplazo de la población y el método de selección. En esta etapa se hace un análisis de los operadores genéticos para observar la forma en que los operadores actúan para resolver el problema con el algoritmo desarrollado.

La finalidad de realizar una gran cantidad de experimentos es establecer los parámetros del algoritmo que le permitan encontrar las mejores soluciones para instancias pequeñas, medianas y grandes, mejor definidas con la instancia Alperovits, Hanoi y Balerma respectivamente. Adicionalmente, dado el carácter estocástico de las heurísticas, las múltiples pruebas experimentales del algoritmo evolutivo realizadas permiten interpretar, con mayor confiabilidad, las tendencias de los resultados obtenidos de manera empírica. Esto derivado de que al repetir los estudios experimentales, para la instancia de prueba Alperovits, se pudo observar discrepancia en los datos resultantes. Con esta observación se llega a la conclusión de que los resultados de 30 ejecuciones del algoritmo, que se presentan generalmente en la literatura, no son suficientes para determinar el comportamiento de un algoritmo, sobre todo cuando éste utiliza constantemente aleatoriedad. Sin embargo, realizar mayor cantidad de pruebas requiere un esfuerzo computacional mayor y sobre todo un esfuerzo considerable para el análisis de los resultados, todo esto aunado a la tendencia de presentar errores, ya que generalmente es un trabajo que se realiza de forma manual.

Con base en esta problemática, el análisis de los resultados de los experimentos realizados en este trabajo, se hace con una herramienta propia, diseñada mediante un procedimiento almacenado en sql y llamado EA-WDNDAnalizer. Esta herramienta ofrece precisión en el análisis de los resultados del algoritmo, ya que opera de manera semi-automática, permitiendo así realizar mayor número de experimentos para aumentar la

confiabilidad de los resultados. Con la herramienta desarrollada se puede hacer un análisis detallado de los resultados obtenidos por los algoritmos desarrollados, permitiendo obtener datos estadísticos importantes como son costos mínimos y máximos de cada generación, costos mínimos y máximos de cada ejecución y promedios de cada generación, todo esto considerando los valores obtenidos en las 30 ejecuciones. Además, el analizador permite obtener promedios generales de las 30 ejecuciones del algoritmo. Esta herramienta tiene la ventaja de que ha sido diseñada especialmente para el análisis de los resultados del algoritmo evolutivo, desarrollado en este trabajo, considerando el formato de los archivos de salida que contienen los resultados de las ejecuciones del algoritmo para evitar trabajo manual e introducción de errores, ya que el analizador funciona automáticamente con solo indicarle la ubicación de los archivos a analizar.

Los archivos que utiliza el analizador como archivos de entrada para el análisis de los resultados son dos: el primero es llamado "*Aptitudes*" + Identificador del proceso que lo ejecutó en el algoritmo evolutivo (por ejemplo Aptitudes0, donde el identificador es un número entero que va desde 0 hasta el número de procesos-1). Este archivo contiene los resultados de los mejores valores en las ejecuciones del algoritmo. Dentro del archivo *Aptitudes* se encuentra un número entero que se refiere a la generación que el algoritmo evolutivo realiza. También contiene un número decimal, que se refiere a la aptitud del mejor individuo encontrado en dicha generación (solución con el menor de los costos).

El archivo *Aptitudes* es utilizado por el analizador EA-WDNDAnalizer para hacer el análisis estadístico, de los resultados obtenidos, de manera eficiente, ya que el archivo sólo contiene información relevante, que es ordenada en cuestión de segundos, para pruebas que contienen de 100 a 1000 generaciones y de 30 ejecuciones del algoritmo.

El segundo archivo, llamado "*Resultados*" + Identificador del proceso, está directamente relacionado con el archivo *Aptitudes* y contiene valores de los parámetros como son: tamaño de población, número de generaciones, probabilidad de cruce y probabilidad de mutación. Así mismo contiene el número de generación, la mejor aptitud, la configuración de la red y los valores de las presiones hidráulicas de la red. Este archivo es muy útil para poder comprobar, en cualquier momento, que la solución encontrada es realmente una solución factible para el problema de Diseño de Redes de Distribución de Agua, ya que al repetir el análisis de la red con el simulador Epanet se pueden corroborar que los resultados hidráulicos que presenta el algoritmo evolutivo son los mismos que

obtiene Epanet en cualquier momento. La estructura de los archivos puede verse en el apéndice II.

Antes de continuar con el análisis de los resultados del algoritmo para ajustar los parámetros, para las diferentes instancias de prueba, es importante definir la interpretación de los resultados, los cuales se encuentran codificados mediante un arreglo de valores reales, véase 5.2.1. Cada una de las soluciones corresponde a una planificación de la red, la cual es una configuración con la definición de un diámetro para cada una de las tuberías de la red y un costo asociado a cada diámetro, lo cual en conjunto determina la calidad de la solución, recordando que el objetivo del problema abordado en esta tesis es minimizar el costo del diseño de una red de distribución de agua, véase 5.3.

El análisis de sensibilidad ha sido realizado sobre tres instancias de prueba clasificadas en la literatura como instancias pequeñas, medianas y grandes, las cuales corresponden a la instancia Alperovits, Hanoi y Balerma definidas previamente en el capítulo 3. La formulación matemática que se aplica a todas instancias de prueba se define en el capítulo 3, mediante el modelo de programación lineal y el modelo de satisfacción de restricciones. Sin embargo, los valores definidos para las constantes de las restricciones 3 y 4 del modelo de satisfacción de restricciones difieren para cada instancia. Así las instancias Alperovits y Hanoi definen valores para H_{min}=30 y H_{max}=100 mientras que Balerma H_{min}=20 y H_{max}=100. Para V_{min}=0.5 mientras que V_{max}=2.5.

6.2.1. Análisis de sensibilidad para la Instancia Alperovits

Los experimentos de ajustes de parámetros realizados para resolver la instancia Alperovits, equivalen a múltiples pruebas independientes, realizadas en diferentes procesadores, con características idénticas. En cada prueba se realizan 30 ejecuciones para observar el comportamiento del algoritmo variando como primer parámetro el tamaño de la muestra (tamaño de la población). Para este experimento se definieron valores iniciales para la probabilidad de cruce y la probabilidad de mutación fijándose en 80 y 20 respectivamente. La condición de paro del algoritmo se estableció fijando el número de generaciones con un valor de 100. Cabe mencionar que las pruebas se iniciaron al mismo tiempo en diferentes procesadores. Cada proceso iniciaba la ejecución del algoritmo, con los parámetros probabilidad de cruce, probabilidad de mutación y número de generaciones fijos. Cada proceso generaba dinámicamente y de forma aleatoria un

tamaño de población que se encontraba dentro de un rango definido, de 100 a 1000 individuos, con incrementos de 100 unidades. Este experimento se realizó una vez y los resultados promedio correspondientes, en unidades monetarias[5], pueden observarse en la tabla 6.1.

Cg_1e_1 +	Cg_1e_2 +	Cg_1e_3 +	… +	Cg_1e_n	= pg_1
Cg_2e_1 +	Cg_2e_2 +	Cg_2e_3 +	… +	Cg_2e_n	= pg_2
Cg_3e_1 +	Cg_3e_2 +	Cg_3e_3 +	… +	Cg_3e_n	= pg_3
.					
Cg_me_{30} +	Cg_me_{30} +	Cg_me_{30} +	… +	Cg_me_n	= pg_n

Figura 6.2 Promedios de las pruebas experimentales

Los promedios finales *p*, mejor representados con la fórmula $p = \sum_{i=1}^{ne} pe$ corresponden a la sumatoria de los promedios de cada ejecución del algoritmo (desde 1 hasta el número de ejecuciones (n_e)), definido en la literatura con el valor de 30. Para obtener *pe* se utiliza la formula donde *pe* es el $pe = \sum_{i=1}^{ng} C\min_i$ promedio de la sumatoria de los valores mínimos de cada generación; n_g es el número de generaciones del algoritmo; C_{min} se refiere al valor mínimo de cada generación, el cual es el costo mínimo del diseño de la red de distribución de agua, Figura 6.3. Esta figura muestra los promedios de las generaciones, desde pg1 hasta pgn. Las pruebas se realizan para diferentes tamaños de la población, recordando que cada prueba se refiere a ejecutar el algoritmo 30 veces, utilizando los mismos parámetros definidos en la primera ejecución del algoritmo. Los resultados de las pruebas se muestran en la tabla 6.1.

[5] Se utiliza unidades monetarias para evitar problemas de conversión a diferentes sistemas monetarios existentes a nivel mundial.

La gráfica de la Figura 6.4 se realizó con el promedio obtenido mediante la fórmula $pg = \sum_{n=1}^{ne} \sum_{m=1}^{ng} C\min_m$ donde pg es el promedio de la sumatoria de los valores mínimos de cada ejecución. Es decir, para m generaciones se realizan n ejecuciones. A menos que se especifique otro valor, generalmente n=30. El promedio se obtiene al sumar los valores mínimos de los costos de la generación cuando m=1, para los valores de n desde 1 hasta 30, para m=2 y n desde 1 hasta 30, y así sucesivamente hasta sumar los costos de los valores mínimos de la generación cuando m=100 y n=30, de acuerdo con la Figura 6.2.

Tabla 6.1 Pruebas Iniciales Instancia Alperovits

Prueba	Tamaño de Población	Promedio (*p*)	Prueba	Tamaño de Población	Promedio (*p*)
1	900	564853	11	400	681600
2	600	468392	12	700	503993
3	600	872337	13	400	499900
4	700	473915	14	900	507252
5	700	840467	15	500	626598
6	500	602078	16	900	736565
7	500	455521	17	900	666836
8	700	669321	18	500	710406
9	200	642243	19	800	481983
10	100	514709	20	700	470379

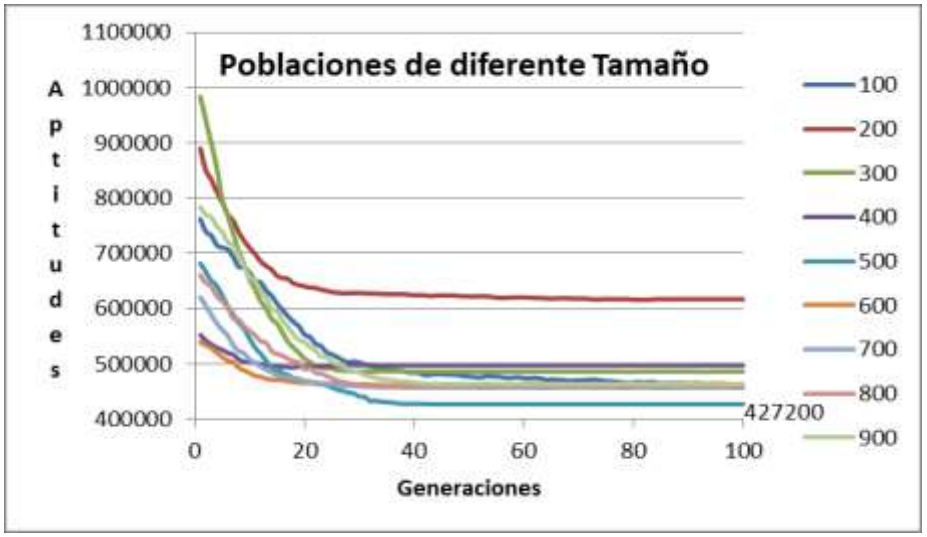


Figura 6.3 Instancia Alperovits con poblaciones de diferente tamaño

Este experimento consistió en definir fijos los parámetros de PCr= 80,PrM=20, NumGen=100, para poblaciones de 100 a 900 individuos. En esta prueba los mejores resultados se obtienen con poblaciones de 500 individuos, encontrando el menor costo para el diseño de la red en 427200 unidades. Sin embargo, para tener mayor confiabilidad en los resultados se realizaron nuevas pruebas, para poblaciones de diferente tamaño. El tamaño de la población se define aleatoriamente en cada procesador. Por ejemplo para población de 100 individuos se realizó únicamente una prueba (10) y se obtuvo como promedio de las 30 ejecuciones 514709 unidades. En el caso de una población de 500 individuos se realizaron 4 pruebas (6, 7, 15 y 18) y se obtuvo como promedio de estas pruebas 598650 unidades, tabla 6. Como puede observarse en la tabla 6, los promedios obtenidos para pruebas de 500 individuos con los mismos parámetros tienen valores diferentes (455521, 602078, 626598, 710406) y esto se debe a la aleatoriedad que maneja constantemente el algoritmo. Se observa que para cualquier tamaño de población los resultados pueden variar considerablemente con cada ejecución del algoritmo. Con base en los resultados de los experimentos, se concluye que 30 ejecuciones del algoritmo no son suficientes para determinar los parámetros adecuados que definan el mejor comportamiento y convergencia del algoritmo. Con la finalidad de realizar un estudio exhaustivo para este trabajo se realizaron 20 pruebas para cada tamaño de población, recordando que cada prueba requiere 30 ejecuciones del algoritmo. Los resultados de las pruebas pueden verse en la Tabla 6.2.

Tabla 6.2 Pruebas para Instancia Alperovits con poblaciones de diferente tamaño

NÚMERO DE PRUEBA	TAMAÑO DE POBLACIÓN								
	100	200	300	400	500	600	700	**800**	900
1	535890	486780	526286	483199	847056	472879	852780	**637756**	509009
2	516621	539937	486607	713214	506264	689512	428765	**484319**	752187
3	609621	707642	487667	470768	497756	480728	462189	**521865**	468752
4	461163	571581	449980	485008	555151	586569	518368	**491372**	642575
5	468849	658093	497763	538769	457807	466439	842611	**508899**	494855
6	502900	515744	488238	480076	822786	896008	549802	**505783**	514102
7	499543	706364	490746	879596	507344	740242	499452	**511462**	458825
8	614557	484578	534024	499914	479244	556007	453117	**638382**	550589
9	538067	512594	492844	504076	532367	532691	701012	**485245**	628409
10	526090	470300	815370	453229	679866	460531	639150	**551247**	480534
11	486304	469556	692765	462359	611474	519209	462197	**501084**	677304
12	846163	545971	602317	535500	489346	724740	572484	**488939**	456082
13	733495	505168	481467	470929	503009	466241	543065	**528106**	442268
14	626049	529191	640443	663293	676836	597409	851966	**491113**	469814
15	560464	465825	510578	524887	542807	654161	695328	**545031**	690299
16	640078	550861	846888	667218	580309	556064	614473	**472631**	520025
17	783135	511757	476219	563010	629546	552699	503238	**476324**	483608
18	530793	471380	534024	656062	529376	453181	497222	**463382**	565835
19	548143	495965	495164	505879	476012	451604	497229	**696715**	468015
20	516621	484697	682759	618946	624578	642354	539820	**544423**	485023
Minimo	419000	419000	419000	419000	419000	419000	419000	419000	419000
Prom.	**577227**	**534199**	**561607**	**558796**	**577447**	**574963**	**586213**	**527204**	**537905**

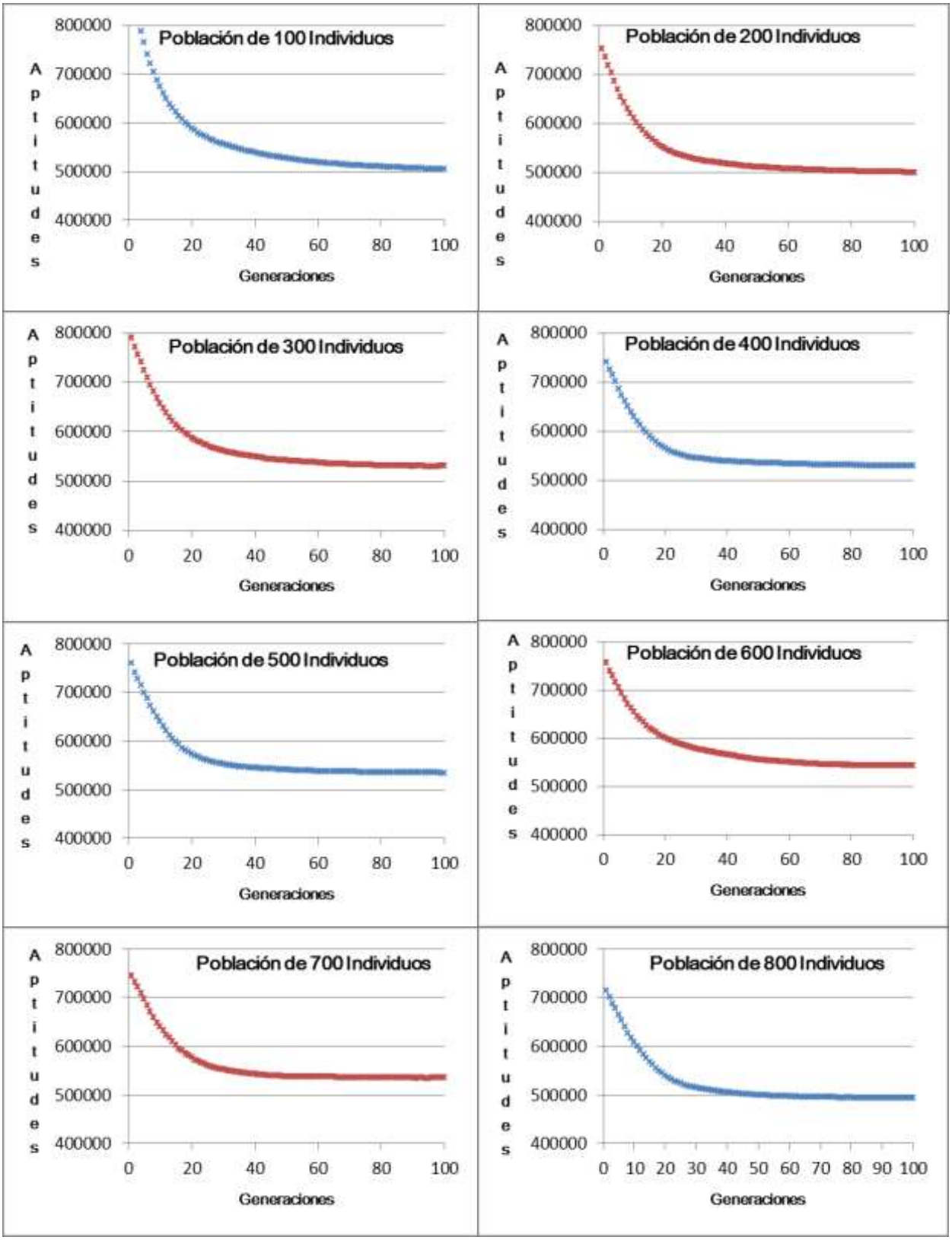


Figura 6.4 Análisis de Sensibilidad de Instancia Alperovits para poblaciones

El tiempo promedio para cada prueba fue de 4 minutos, gracias a que la instancia Alperovits es una instancia considerada como pequeña. En la tabla 6.4 puede observarse que los costos promedio finales (Prom.) de las pruebas son cercanos. Es importante mencionar que el algoritmo EA-WDND encuentra la mejor cota conocida con el valor de 419,000 unidades, en todas las pruebas realizadas. Al igual que la tabla 6.2, las gráficas de la Figura 6.5 muestran que los mejores resultados promedio se obtienen con poblaciones de 800 individuos. Se observa en la gráfica que el costo del diseño de la red disminuyó considerablemente de 800,000 a 500,000 en las primeras 30 generaciones. De acuerdo con los resultados de las pruebas realizadas para el tamaño de población se puede concluir que lo más adecuado para el algoritmo es trabajar con poblaciones de 800 individuos, ya que con este tamaño de población se obtienen los menores costos promedio para el diseño de la red de distribución de agua, para la instancia Alperovits, los cuales son cercanos a 500,000 unidades monetarias, Figura 6.5.

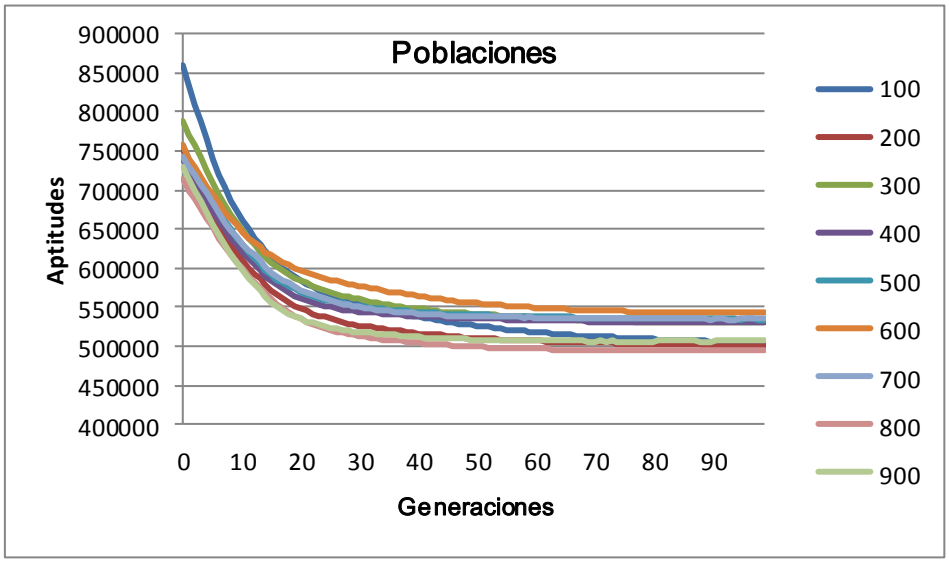


Figura 6.5 Análisis de Sensibilidad para sintonizar población

Una vez que se define el tamaño de poblaciones a utilizar en 800 individuos, el siguiente experimento consistió en realizar pruebas con el número de generaciones para determinar la convergencia del algoritmo. De acuerdo con los resultados de las pruebas realizadas, para el número de generaciones, se puede concluir que lo más adecuado para el algoritmo es trabajar con 800 generaciones, aunque la gráfica muestra que el algoritmo converge a partir de la generación 20, para la instancia Alperovits, Figura 6.6. Finalmente,

se realizó un experimento que consistió en realizar pruebas con las probabilidades de cruce y de mutación, que prácticamente se refieren al número de individuos de la población, que serán combinados para generar descendientes y el número de individuos que sufrirán alguna mutación convirtiéndose en nuevos individuos. Este experimento se realizó con poblaciones de 800 individuos, los cuales representan el 100% de la muestra.

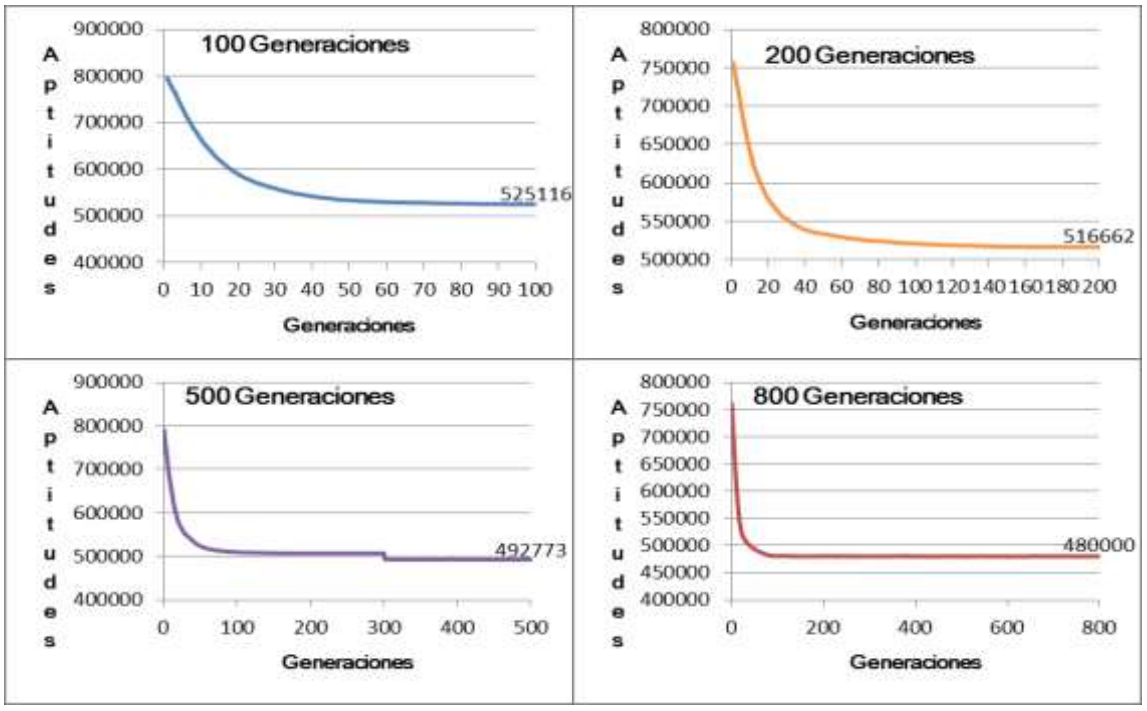


Figura 6.6 Sintonizando Generaciones

El procedimiento para definir la probabilidad de cruce se hace con base en los estudios definidos en la literatura que muestran que el intervalo de probabilidad de cruce, para el funcionamiento del algoritmo evolutivo, se encuentra generalmente entre 60 y 90%. Así para establecer la probabilidad de cruce (PrCr) se elige de forma aleatoria un número comprendido en dicho intervalo, PrCr = Aleatorio (60,90), Tabla 6.7. La probabilidad de mutación (PrMt), definida en el algoritmo SEA-WDND, depende de la probabilidad de cruce y se obtiene mediante la fórmula PrMt =100%- PrCr. La probabilidad de mutación se obtiene restando al 100% (que representa el total de la muestra) el valor definido para la probabilidad de cruce. De acuerdo con los resultados de las pruebas, Figura 6.8, se observa que el algoritmo se comporta mejor cuando las probabilidades de

cruce son de 80 a 90 y las probabilidades de mutación son de 10 a 20 y la justificación de este comportamiento se explica a continuación: de acuerdo con el estudio de cruce de individuos factibles realizado, se sabe que al combinar dos individuos factibles, existe 90% de probabilidad de obtener un descendiente factible. Así mismo, se sabe que si se muta un individuo factible, con un operador de mutación de 0.03, la probabilidad de que el nuevo individuo sea factible es de aproximadamente 70%. Por lo que en una muestra de 100 individuos si 80 de ellos se combinan, se sabe que en el mejor de los casos 144 de ellos serán descendientes factibles. Al mutar el 20% restante, con una probabilidad de 3%, se tendrán, en el mejor de los casos, a 28 individuos factibles. Así, de la muestra de 100 individuos se tendrá como resultado una población de 172 descendientes factibles en el mejor de los casos.

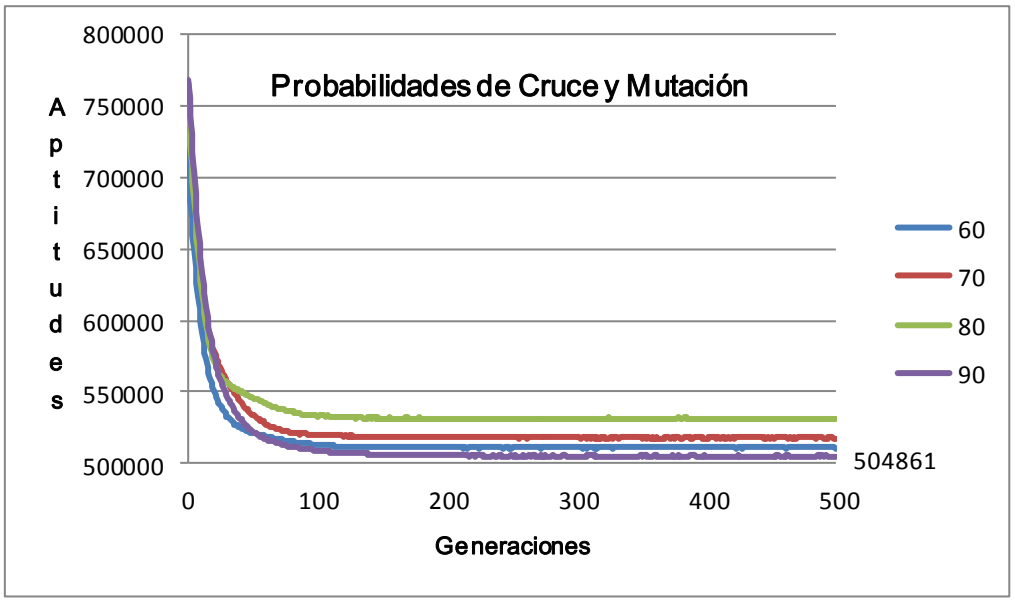


Figura 6.7 Probabilidad de Cruce y Mutación

Para el escenario en el que la probabilidad de cruce es de 60%, la probabilidad de mutación queda definida con el valor de 40% que indica que de los 800 individuos a lo sumo 320 sufrirán alguna mutación. Al definir probabilidad de cruce en 60% se asume que en el mejor de los casos 480 individuos de la población (de 800 individuos) serán combinados, con lo cual se generarán 960 descendientes, de los cuales aproximadamente **864** serán individuos factibles (Descendientes Cruce) y 800 de ellos podrán formar una nueva población. En cuanto a la mutación, considerando que 70% de

ellos serán factibles se obtienen **224** nuevos individuos (individuos mutados) que pueden formar parte de una nueva población en la siguiente generación. Así, en este escenario, se obtienen 1088 nuevos individuos de los cuales 800 de ellos serán seleccionados para formar parte de una nueva población, dependiendo directamente la aptitud que tengan o del método de selección que se utilice para el remplazo. Cabe mencionar que de la población de 800 individuos algunos de ellos pueden combinarse y generar descendientes e incluso, también pudieran sufrir alguna mutación y convertirse en nuevos individuos. De esta forma, es como se genera una nueva población de individuos, en el algoritmo evolutivo EA-WDND. La Tabla 6.3 muestra los resultados de aplicar, a una población de 800 individuos, las probabilidades de cruce con valores de 60 a 90%.

Tabla 6.3 Probabilidad de Cruce y Mutación para población de 800 individuos

Pr.Cr	Pr.Mt	Descendientes Cruce	Descendientes Mutación	Total Individuos
60	40	864	224	1088
70	30	1008	168	1176
80	20	1152	112	1264
90	10	1296	56	1352

Tabla 6.4 Pruebas de sintonización para Instancia Alperovits

	Probabilidad de Cruce			
Individuos	**60**	**70**	**80**	**90**
100	136	147	158	169
200	272	294	316	338
300	408	441	474	507
400	544	588	632	676
500	680	735	790	845
600	816	882	948	1014
700	952	1029	1106	1183
800	1088	1176	1264	1352
900	1224	1323	1422	1521
1000	1360	1470	1580	1690

La conclusión final de este estudio, Tabla 6.4, fue que con probabilidades de cruce de 80 a 90, el crecimiento de la población es mayor que con probabilidades de cruce de 60 o 70. Con ello, se tiene una muestra mayor de la cual se pueden seleccionar a los mejores descendientes para formar una nueva población, obteniendo a través del tiempo poblaciones con mejores individuos que evolucionan en cada generación. Con los resultados de estos experimentos de sintonización de parámetros, para instancias de tamaño pequeño como Alperovits, se concluye que los mejores resultados del algoritmo se obtienen con poblaciones de 800 individuos, realizando 800 generaciones con una probabilidad de cruce de 90% y de mutación de 10%. Es importante decir que en todas las pruebas realizadas, con los valores definidos, el algoritmo evolutivo secuencial encuentra la mejor cota conocida en la literatura, la cual tiene un costo de 419,000 unidades monetarias para el diseño de la red de distribución de agua Alperovits y en promedio el tiempo de ejecución de 30 pruebas del algoritmo es de aproximadamente 4 minutos.

6.2.2. Resultados de los experimentos para Instancia Alperovits

En este trabajo de tesis se obtuvieron resultados de experimentos con diferentes instancias de prueba teóricas, utilizadas frecuentemente por varios investigadores, con la finalidad de comparar los resultados obtenidos por el algoritmo evolutivo EA-WDND. Además de los resultados mostrados en el análisis de sensibilidad previo, en este apartado se muestran resultados de los experimentos en cuanto al tiempo requerido para la ejecución del algoritmo. Es importante decir, que en la mayoría de los trabajos con los que se compara el algoritmo no se menciona el tiempo que éste requiere para la solución del problema. En este caso, se puede comparar el algoritmo desarrollado con otros trabajos únicamente en cuanto a la calidad de las soluciones que encuentran.

El algoritmo Evolutivo secuencial encuentra la mejor cota conocida para la función de aptitud para la instancia Alperovits, que es un costo de 419,000 unidades monetarias. Se realizaron pruebas para la red Alperovits considerando las restricciones de velocidades y sin considerarlas y los resultados obtenidos fueron similares (ligeramente mayores costos cuando se utiliza la restricción de velocidades). En ambos casos el

algoritmo evolutivo secuencial encuentra la mejor cota conocida. Sin embargo, es importante mencionar que en estos estudios experimentales se comprueba que al aumentar el tamaño de la población y el número de generaciones, el tiempo requerido para la ejecución del algoritmo se incrementa de manera exponencial, tal como lo muestra la gráfica de la Figura 6.9.

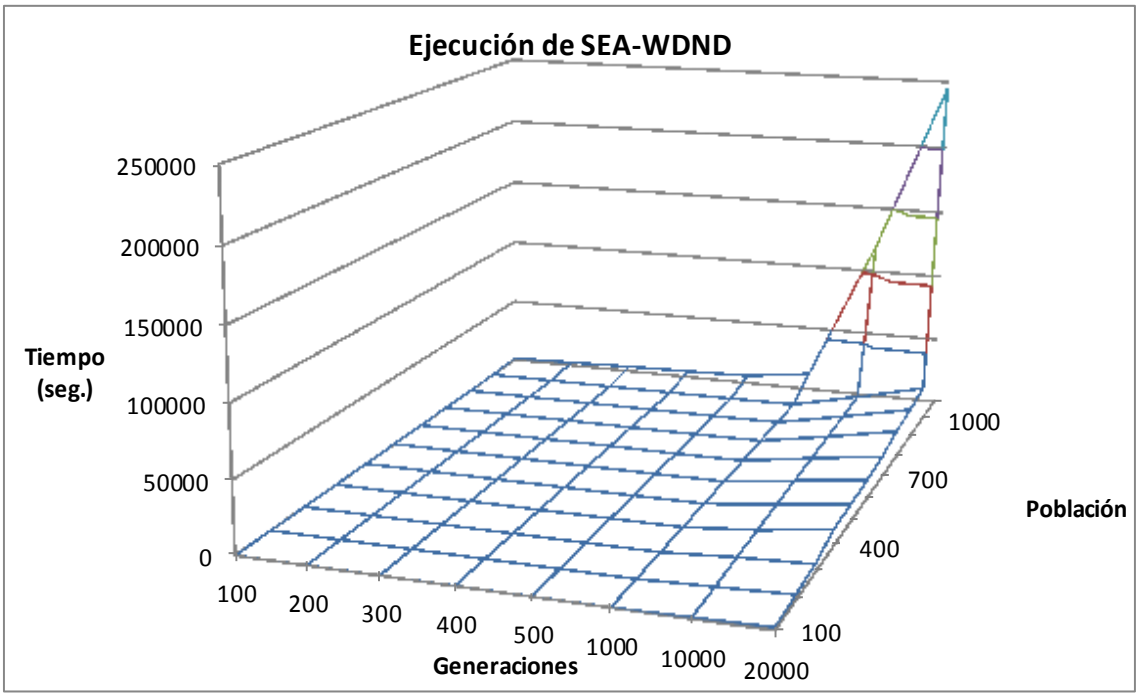


Figura 6.8 Tiempos de ejecución Instancia Alperovits

6.2.3. Análisis de sensibilidad para la Instancia Hanoi

El algoritmo EA-WDND, desarrollado en este trabajo, tiene implementadas dos funciones diferentes para la creación de la población inicial. Para instancias pequeñas, como Alperovits, es suficiente con la creación de la población inicial aleatoria. Pero para las instancias de prueba, medianas y grandes (Hanoi y Balerma respectivamente), la creación de la población inicial se hace de forma determinista mediante una búsqueda local iterada. Para instancias medianas y grandes el algoritmo usa la función *AjustaaFactibles,* la cual es una función de búsqueda local iterada. Esta función no era indispensable al resolver instancias pequeñas ya que el grado de dificultad para generar soluciones

factibles es relativamente bajo y por esa razón no resultaba importante convertir soluciones infactibles a factibles. No obstante, cuando se resuelven instancias medianas y grandes, es sumamente importante conservar soluciones cuasi-factibles, ya que desecharlas y volver a crear factibles realmente representa un alto costo computacional que la mayoría de los algoritmos evolutivos presentan. El experimento de sintonización realizado para resolver la instancia Hanoi consistió en la generación de múltiples pruebas, cada prueba consistió en 30 ejecuciones independientes del algoritmo. El proceso de análisis de sensibilidad es similar para todas las instancias resueltas en este trabajo de tesis. Pero para instancias medianas como la instancia Hanoi se realiza un número mayor de generaciones en el Algoritmo Evolutivo, con la finalidad de encontrar la mejor solución. Así mismo, se manejan también poblaciones de diferente tamaño. En concreto, las poblaciones que se definen para la instancia Hanoi son de 100, 500, 1,000, 5000 y 10,000 individuos. El número de generaciones para esta instancia establece valores de 100, 500, 1,000, 5000, 10,000 y 20,000 iteraciones, todo esto con la finalidad de observar el comportamiento del algoritmo cuando se definen los rangos de valores más amplios para los parámetros del algoritmo.

Una vez definidos los valores paramétricos del algoritmo, se realizan 30 ejecuciones del algoritmo variando como primer parámetro el tamaño de la población para observar su comportamiento. Para este experimento se definieron valores iniciales para la probabilidad de cruce, la probabilidad de mutación y el número de generaciones, fijándose en 80, 20 y 100 respectivamente. La creación de la población inicial para la instancia Hanoi, consistió en un proceso de dos etapas. En la primera etapa se crea un individuo factible de forma determinista, mediante la asignación del mayor de los diámetros disponibles a todas las tuberías que forman la configuración de la red. Al hacer este proceso se verifica que la configuración cumpla las restricciones del modelo hidráulico, observando que las presiones máximas encontradas son de 96 mca. Posteriormente, en una segunda etapa se aplica una serie de cambios a la configuración definida, con la finalidad de obtener diferentes individuos en la población que puedan combinarse y generar descendientes que continúan la evolución de la población. Los escenarios de cambios de diámetros de un individuo son tres y cualquiera de éstos puede ocurrir con la misma probabilidad. Es decir, para cambiar cada tubería de la configuración definida se tienen tres opciones 1) disminuir diámetro, 2) elegir diámetro de la primera mitad de

diámetros disponibles, los cuales son los diámetros más pequeños y 3) elegir diámetro de la segunda mitad de diámetros disponibles, que corresponden a los diámetros de mayor tamaño. Este experimento se realizó con una muestra de 150,000 individuos y los resultados se muestran en la Figura 6.9. En esta Figura se observa que entre mayor sea el número de cambios realizados a la configuración, menor es el número de individuos factibles resultantes. Incluso, se puede observar que cuando se realizan cambios genéticos en todo el cromosoma la cantidad de individuos factibles tiende a acercarse a cero. Este estudio justifica la dificultad de crear una población inicial factible de forma aleatoria para instancias medianas, ya que entre mayor número de cambios ocurran en un individuo mayor es la probabilidad de que éste sea infactible.

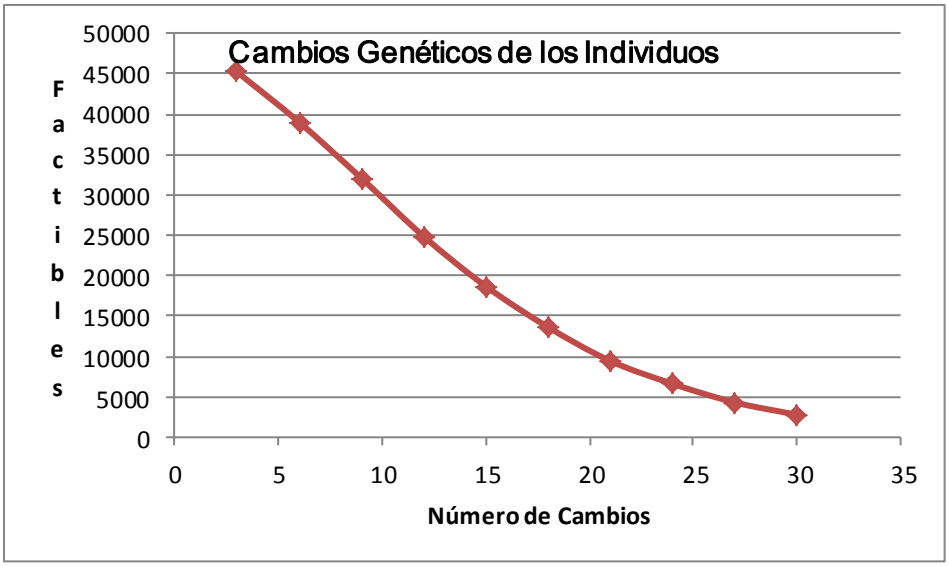


Figura 6.9 Creación de población factible para instancias medianas

Un experimento adicional antes de iniciar con la sintonización del algoritmo para instancias medianas, consistió en realizar pruebas para observar el comportamiento del algoritmo cuando se utiliza en él la función de *AjustaaFactible*, la cual se ha explicado a detalle en el algoritmo 5.9.El experimento consistió en elegir una población de 500 individuos y probar el funcionamiento del algoritmo evolutivo en dos casos: 1) utilizando la función *AjustaaFactible* y 2) sin utilizar la función *AjustaaFactible*. Los resultados promedio de 30 ejecuciones del algoritmo se muestran en las gráficas de la Figura 6.10. Con los resultados se concluye que el algoritmo evolutivo obtiene mejores resultados cuando utiliza la función *AjustaaFactible*. Este hecho está fundamentado en que los

resultados muestran que el costo promedio mínimo encontrado cuando no se utiliza dicha función es de 6330090 mientras que el costo promedio resultante al usar la función *AjustaaFactible* es de 6315585. El siguiente procedimiento una vez que se determinó utilizar la función *AjustaaFactible* en el algoritmo evolutivo SEA-WDND, consistió en realizar el análisis de sensibilidad para encontrar los mejores valores paramétricos del algoritmo para instancias medianas como lo es la instancia Hanoi.

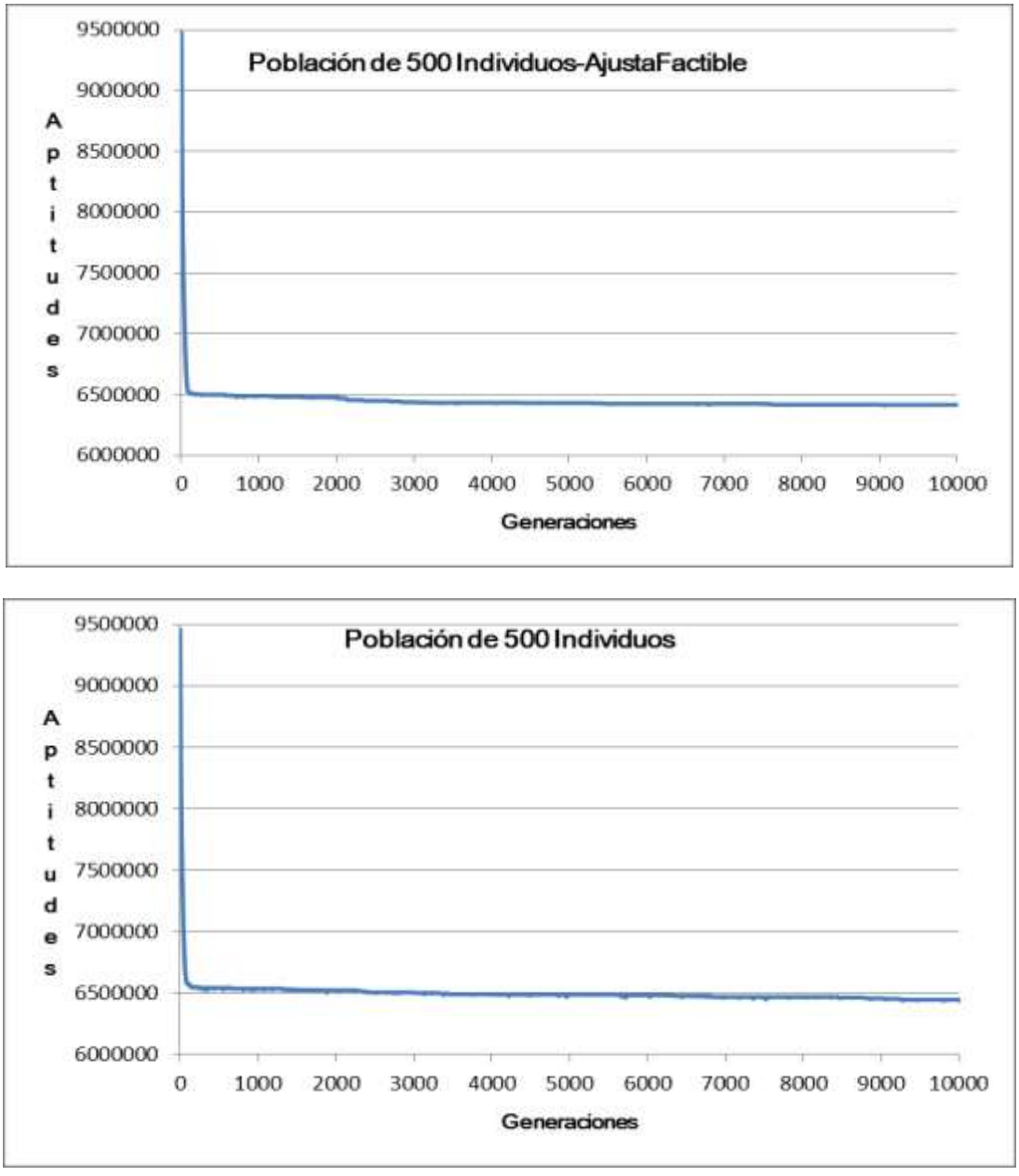


Figura 6.10 Creación de población factible para instancias medianas

Para el ajuste paramétrico, se definen poblaciones de 100, 500, 1,000, 5,000 y 10,000 individuos y, para cada una de ellas, se hacen pruebas con 100, 500, 1000, 5,000, 10,000 y 20000 generaciones. Cabe mencionar, que para la sintonización del algoritmo, se considera como criterio de paro del algoritmo el número de generaciones. La finalidad de este estudio es observar la convergencia y comportamiento del algoritmo. En este sentido, el tiempo que tarde en la ejecución del algoritmo es irrelevante ya que el principal objetivo es observar la evolución de las poblaciones para obtener los mejores resultados. Es importante decir que en la práctica la ejecución del algoritmo debe terminarse en cuestión de segundos o máximo minutos, ya que en problemas reales de optimización el tiempo de ejecución de un algoritmo es un factor realmente importante.

Tabla 6.5 Pruebas para 100 Individuos

		GENERACIONES					
		100	**500**	**1000**	**5000**	**10000**	**20000**
NÚMERO DE PRUEBA	**1**	6684755	6613214	6562382	6593025	6508814	6416006
	2	6860507	6669427	6562382	6426140	6482651	6416006
	3	7017086	6786100	6344086	6545872	6609179	6654593
	4	6849685	6586366	6595361	6452820	6493233	6678206
	5	6684755	6727562	6512388	6516988	6466136	6570067
	6	6786780	6527576	6534782	6687227	6533124	6500908
	7	6860507	6601585	6595361	6404079	6591709	6500908
	8	6849685	6868528	6556819	6437977	6503226	6523438
	9	6684755	6620330	6650791	6369359	6422259	6556829
	10	6863255	6547330	6741790	6407448	6543349	6444108
	11	6961838	6705430	6783214	6524410	6564247	6561399
	12	6808306	6716002	6606718	6485847	6470554	6528992
	13	7179257	6695621	6600431	6394687	6478424	6430128
	14	6849453	6881282	6529432	6392327	6596077	6651351
	15	6865108	6881282	6546350	6392327	6614797	6510719
	16	6699730	6538983	6585879	6450080	6717424	6510719
	17	6889482	6593558	6834210	6591047	6368189	6468129
	18	6822318	6791460	6569641	6490065	6423187	6682380
	19	7196330	6989363	6534782	6535316	6517975	6402001
	20	6808306	7187265	6834210	6437977	6482651	6731938
Min.		6684755	6527576	6344086	6369359	6368189	6402001
Prom.		6861095	6726413	6604050	6476751	6519360	6536941

La tabla 6.5 muestra los resultados del algoritmo con una población de 100 individuos. Con estos resultados se concluye que el promedio mínimo de las pruebas se obtiene con 5,000 generaciones, con un costo de 6476751, y un costo mínimo de la red con un valor de 6369359.

Tabla 6.6 Pruebas para 500 Individuos.

	GENERACIONES					
NÚMERO DE PRUEBA	100	500	1000	5000	10000	20000
1	6572090	6467518	6562382	6410757	6418710	6366800
2	6605026	6469071	6562382	6419679	6315586	6341100
3	6605026	6422768	6344086	6384816	6463558	6546065
4	6446256	6553565	6466713	6387286	6340023	6452157
5	6511822	6480370	6445916	6477731	6376765	6442655
6	6556526	6500775	6482336	6570320	6383244	6369092
7	6551298	6523711	6482336	6417895	6368232	6310752
8	6497695	6482591	6472522	6498967	6496763	6326716
9	6515121	6512618	6412890	6505004	6496763	6498583
10	6581658	6494143	6508230	6323499	6397569	6361919
11	6536694	6665422	6404227	6406509	6388465	6307532
12	6552372	6345459	6534782	6328622	6509674	6304917
13	6607517	6416655	6518124	6397655	6395012	6308704
14	6501755	6524391	6457316	6383884	6327926	6452087
15	6561569	6474087	6512388	6413953	6491152	6454828
16	6456436	6380594	6512388	6460629	6352967	6377490
17	6545244	6498526	6539601	6456771	6337336	6325190
18	6527693	6558442	6595361	6420202	6468190	6352335
19	6502735	6500226	6595361	6504720	6422280	6467904
20	6520286	6493215	6344086	6486033	6368232	6440038
Min.	6446256	6345459	6344086	6323499	6315586	6304917
Prom.	6537741	6488207	6487671	6432746	6405922	6390343

Las tablas 6.5 a 6.8 presentan los resultados promedio de las pruebas experimentales realizadas. En todos los experimentos, excepto las pruebas con poblaciones de 10,000 individuos, se realizaron 20 pruebas, cada prueba consistió en 30

ejecuciones del Algoritmo Evolutivo. En el caso de poblaciones con 10,000 individuos se realizaron 14 pruebas, tabla 6.9, debido principalmente al tiempo que se requiere para la finalización de las mismas. Los resultados que se muestran en las tablas son los valores mínimos encontrados en cada ejecución del algoritmo. También se muestran en la última fila los promedios obtenidos de los valores mínimos encontrados.

Tabla 6.7 Pruebas para 1000 Individuos

		GENERACIONES					
		100	**500**	**1000**	**5000**	**10000**	**20000**
NÚMERO DE PRUEBA	**1**	6572090	6461231	6468979	6453222	6431245	6444536
	2	6605026	6418276	6414665	6297300	6431245	6444536
	3	6605026	6417458	6403265	6452087	6341269	6454297
	4	6446256	6533254	6519683	6492273	6425416	6454297
	5	6511822	6420220	6425473	6492273	6397232	6344970
	6	6556526	6441460	6403410	6377693	6466584	6326544
	7	6551298	6376836	6466577	6444713	6370947	6326544
	8	6497695	6350395	6408279	6315099	6361842	6454297
	9	6515121	6442532	6408279	6315099	6314136	6362912
	10	6581658	6533254	6348891	6452087	6305921	6339820
	11	6536694	6533254	6457339	6483835	6360547	6344087
	12	6552372	6420220	6484446	6483835	6386865	6380687
	13	6607517	6372843	6457979	6380174	6378520	6330555
	14	6501755	6441460	6472366	6452087	6378529	6336073
	15	6561569	6396683	6472366	6457570	6360082	6329112
	16	6456436	6376836	6395012	6457570	6357661	6329112
	17	6545244	6350395	6435706	6387755	6324172	6466684
	18	6527693	6350395	6435706	6444741	6337946	6466684
	19	6502735	6393038	6408076	6369460	6338055	6442803
	20	6520286	6442532	6481540	6336801	6309789	6326544
Min.		6446256	6350395	6348891	6297300	6305921	6326544
Prom.		6537741	6423628	6438402	6417283	6368900	6385254

La tabla 6.6 muestra los resultados del algoritmo con una población de 500 individuos. Con estos resultados se concluye que el promedio mínimo de las pruebas se

obtiene con 20,000 generaciones, con un costo de 6390343. El costo mínimo de la red tiene un valor de 6304917. La tabla 6.7 muestra los resultados del algoritmo con una población de 1000 individuos. Con estos resultados se concluye que el promedio mínimo de las pruebas se obtiene con 10,000 generaciones, con un costo de 6368900. El costo mínimo de la red tiene un valor de 6297300.

Tabla 6.8 Pruebas para 5000 Individuos

	GENERACIONES						
		100	500	1000	5000	10000	20000
NÚMERO DE PRUEBA	1	6345616	6387099	6334227	6336910	6372137	6432418
	2	6345616	6403471	6343560	6336910	6372486	6370111
	3	6416819	6399611	6343560	6307532	6373439	6365263
	4	6362160	6356427	6355759	6338904	6372137	6432418
	5	6402589	6356427	6355759	6341100	6372486	6364564
	6	6398445	6354513	6374429	6309790	6373439	6369372
	7	6361842	6354513	6374429	6330909	6372137	6311441
	8	6378197	6450213	6366689	6442084	6372486	6373977
	9	6378197	6452087	6366689	6442084	6373439	6369081
	10	6380687	6352363	6332418	6309790	6372137	6369913
	11	6395794	6454670	6342865	6294141	6372486	6369081
	12	6389948	6373199	6454297	6340413	6373439	6369333
	13	6402289	6346946	6343134	6337778	6372486	6368986
	14	6389948	6385902	6347455	6326388	6373439	6362434
	15	6393926	6365857	6357195	6341181	6372137	6316212
	16	6385933	6394586	6357195	6309790	6372486	6369081
	17	6385933	6366189	6340811	6294141	6373439	6368986
	18	6402288	6364863	6351486	6340413	6372137	6316212
	19	6394670	6452087	6347562	6338904	6372486	6311441
	20	6395939	6348016	6343560	6341100	6373439	6373977
Min.		6345616	6346946	6332418	6294141	6372137	6311441
Prom.		6385342	6385952	6356654	6338013	6372715	6364215

La tabla 6.8 muestra los resultados del algoritmo con una población de 5000 individuos. Con estos resultados se concluye que el promedio mínimo de las pruebas se

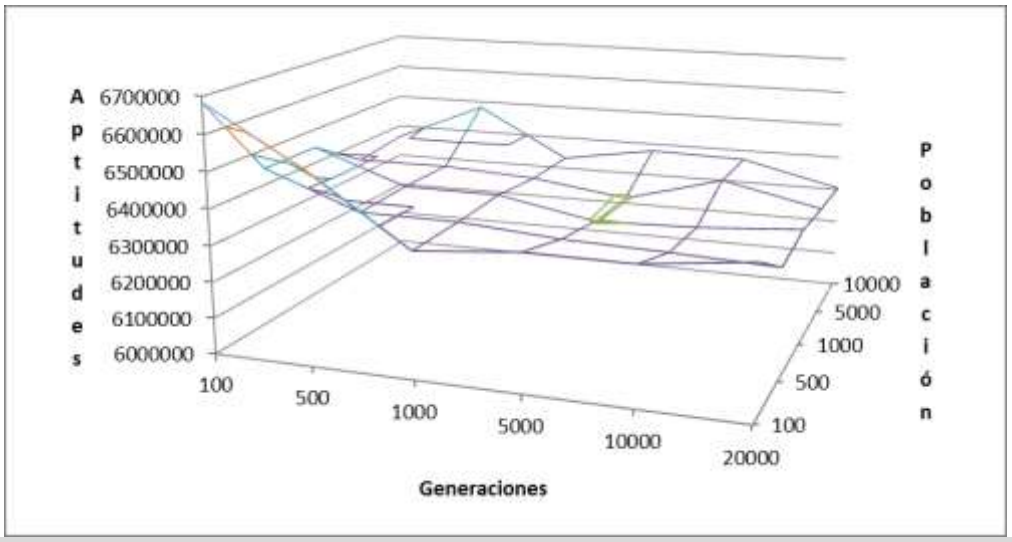
obtiene con 5,000 generaciones, con un costo de 6338013. El costo mínimo de la red tiene un valor de 6294141. En la tabla 6.9, se muestran las pruebas experimentales que se realizaron con poblaciones de 10,000 individuos. En estas pruebas se concluye que los mejores resultados se obtienen con 1000 generaciones, obteniendo promedio mínimo de 6348605 y costo mínimo de 6332904.

Tabla 6.9 Pruebas para 10,000 Individuos

		GENERACIONES				
		100	**500**	**1000**	**5000**	**10000**
NÚMERO DE PRUEBA	**1**	6387099	6500937	6356446	6379785	6372234
	2	6385601	6503214	6341100	6377975	6372684
	3	6364343	6481476	6332904	6379731	6371882
	4	6391025	6486223	6355760	6382874	6371767
	5	6378529	6486223	6356156	6378357	6372158
	6	6378197	6495923	6336515	6379153	6372680
	7	6371702	6500066	6334349	6377957	6371966
	8	6367355	6491742	6361014	6381142	6378511
	9	6374697	6491981	6336916	6382874	6371882
	10	6378529	6486320	6357813	6378357	6371767
	11	6391446	6487877	6355760	6379153	6372158
	12	6388535	6504118	6354715	6377957	6372680
	13	6388535	6504118	6345724	6381142	6371966
	14	6388535	6483706	6355301	6379785	6378511
Min.		6364343	6481476	6332904	6377957	6371767
Prom.		6381009	6493137	6348605	6379732	6373061

La Figura 6.11 muestra un resumen de los resultados de las pruebas experimentales previamente descritas. En esta tabla se presentan los promedios mínimos obtenidos para diferentes tamaños de poblaciones y diferentes números de generaciones. En esta tabla, puede observarse que los mejores resultados de las pruebas experimentales se obtienen con poblaciones de 5000 individuos y 5000 generaciones. Los valores mínimos obtenidos por el algoritmo para diferentes tamaños de poblaciones y diferentes números de generaciones se muestran en la Figura 6.12. En conclusión se

observa de los resultados que los mejores valores, tanto en promedio como en costos mínimos, se obtienen con poblaciones de 5000 individuos y 5000 generaciones.



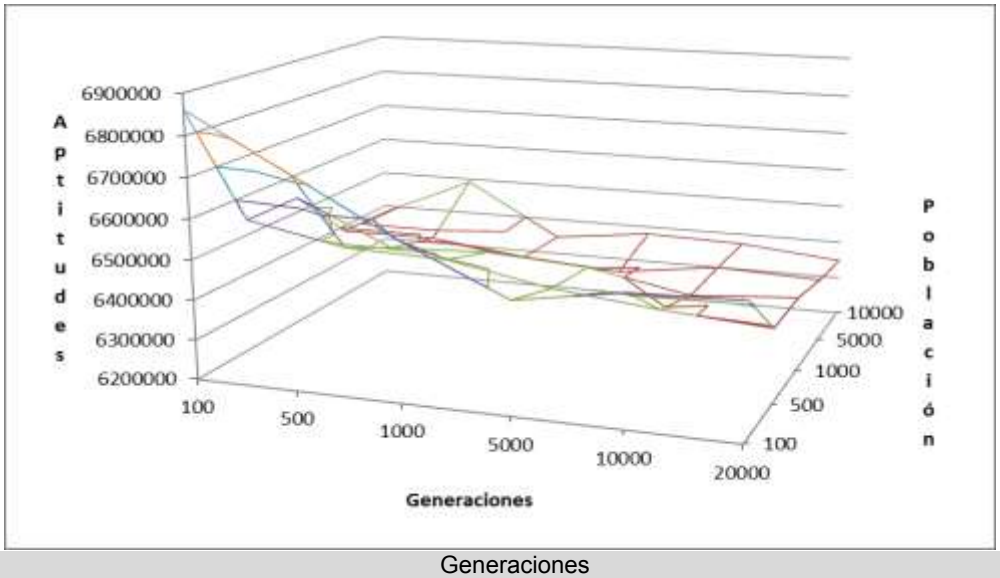
Generaciones

Tamaño Población	100	500	1000	5000	10000	20000
100	6861095	6726413	6591671	6476751	6519360	6526678
500	6537741	6488207	6480834	6432746	6405922	6387727
1000	6537741	6423628	6434139	6417283	6368900	6388344
5000	6385342	6385952	6355500	**6338013**	6373061	6363701
10000	6380941	6491591	6348605	6379732	6373061	6348334

Figura 6.11 Costos Promedio de Pruebas Iniciales Instancia Hanoi

Las gráficas de las Figuras 6.13 a 6.18 muestran los resultados gráficos de los resultados previamente descritos en las tablas, para poblaciones de 100, 500, 1000, 5000 y 10,000 individuos y con diferente número de generaciones. Las gráficas de dichas Figuras muestran la convergencia del algoritmo SEA-WDND para la instancia de prueba Hanoi. En estas gráficas se observa que el algoritmo converge en las primeras 100 generaciones.

Con las pruebas experimentales realizadas se decide fijar el tamaño de la población y el número de generaciones en 5,000 porque con estos valores se obtuvieron los mejores resultados. Posteriormente se procede a realizar las pruebas con los operadores genéticos de probabilidad de cruce y probabilidad de mutación. Para la probabilidad de cruce se asignan valores de 50 a 100 con incrementos de 10 unidades y se observa que los mejores resultados de las pruebas se obtienen cuando la probabilidad de cruce tiene el valor de 80. Para la probabilidad de mutación se asignan valores de 10 a 50 con incrementos de 10 unidades.



Tamaño Población \ Generaciones	100	500	1000	5000	10000	20000
100	6684755	6527576	6344086	6369359	6368189	6402001
500	6446256	6345459	6344086	6323499	6315586	6304917
1000	6446256	6350395	6348891	6297300	6305921	6326544
5000	6345616	6346946	6332418	**6294141**	6371767	6311441
10000	6364343	6481476	6332904	6377957	6371767	6304814

Figura 6.12 Costos Mínimos de Pruebas Iniciales Instancia Hanoi

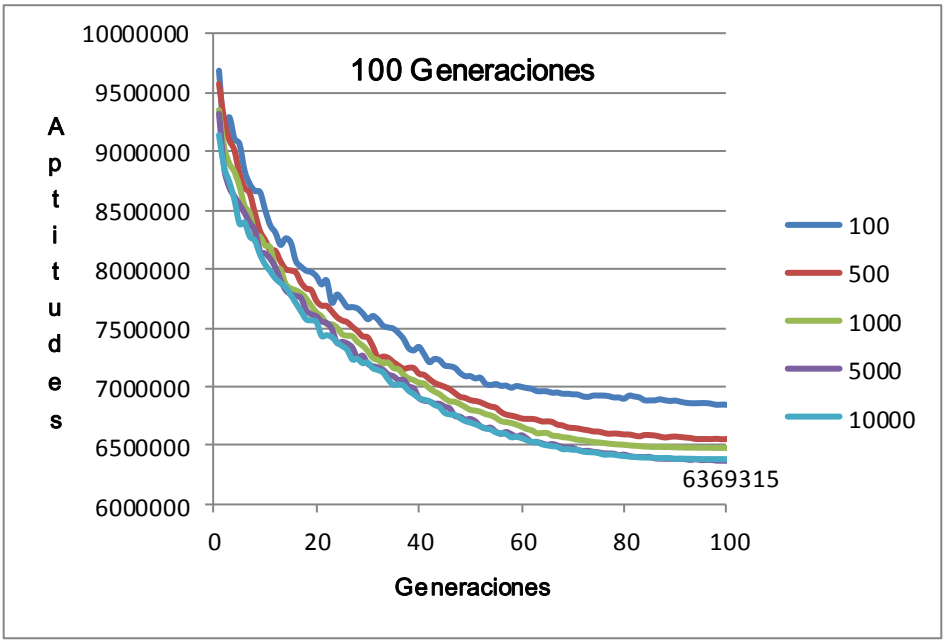


Figura 6.13 Cien Generaciones con Poblaciones de Diferente Tamaño

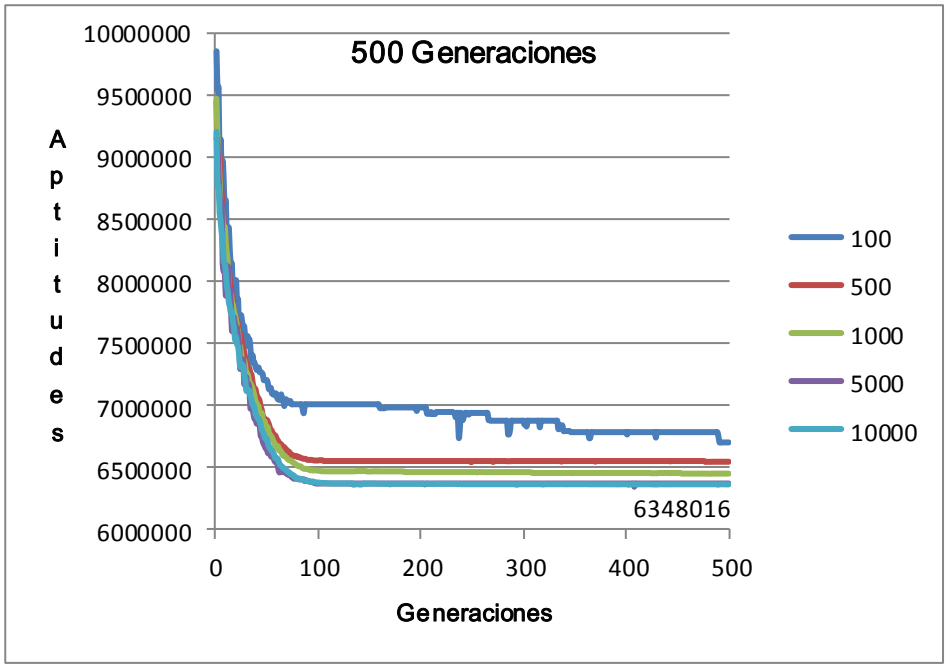


Figura 6.14 Quinientas Generaciones con Poblaciones de Diferente Tamaño

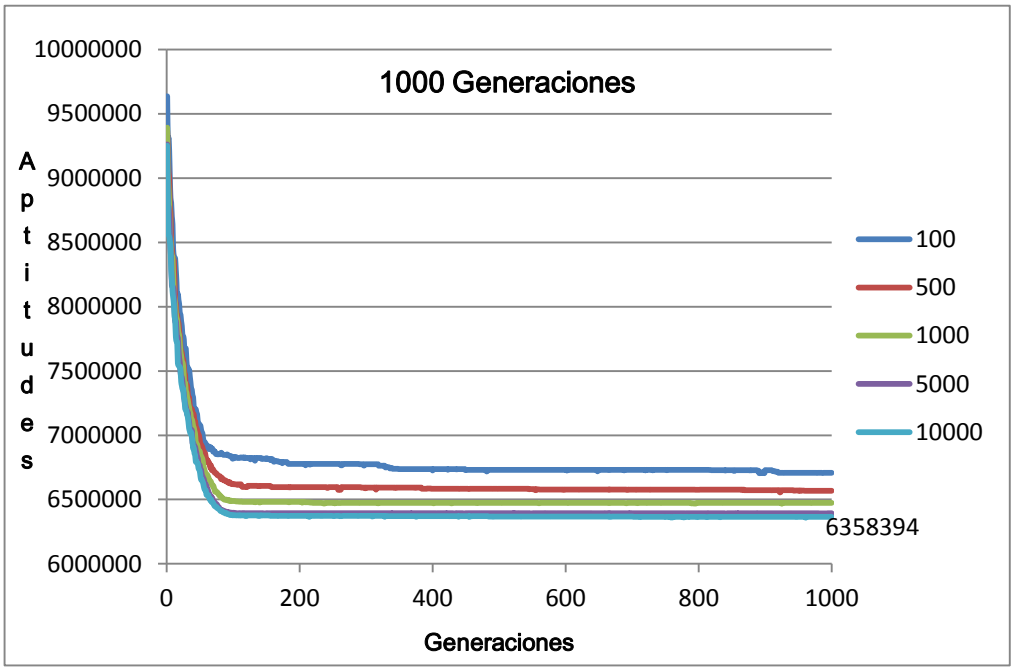


Figura 6.15 Mil Generaciones con Poblaciones de Diferente Tamaño

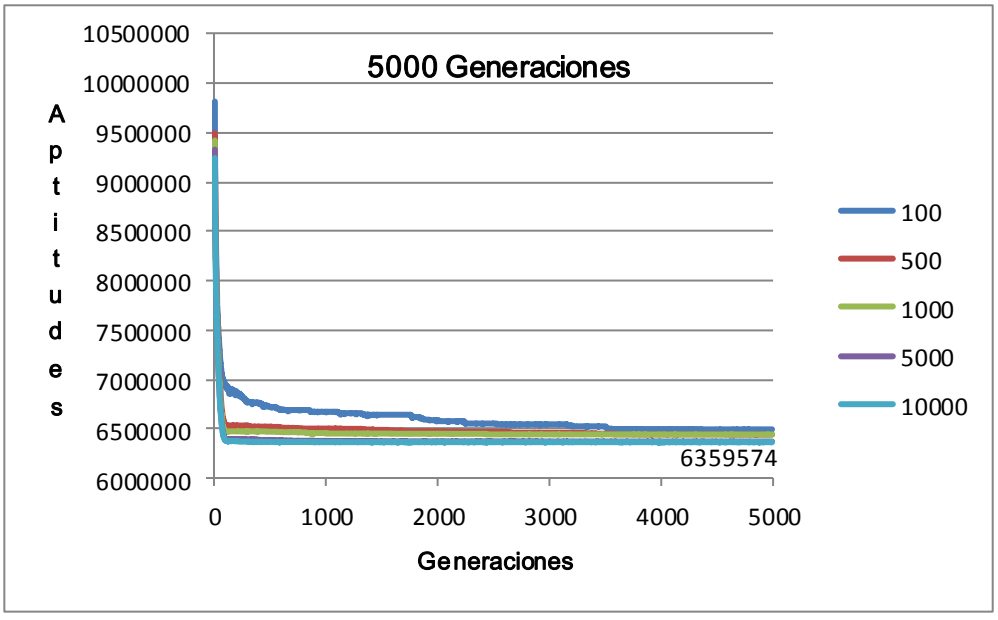


Figura 6.16 Cinco Mil Generaciones con Poblaciones de Diferente Tamaño

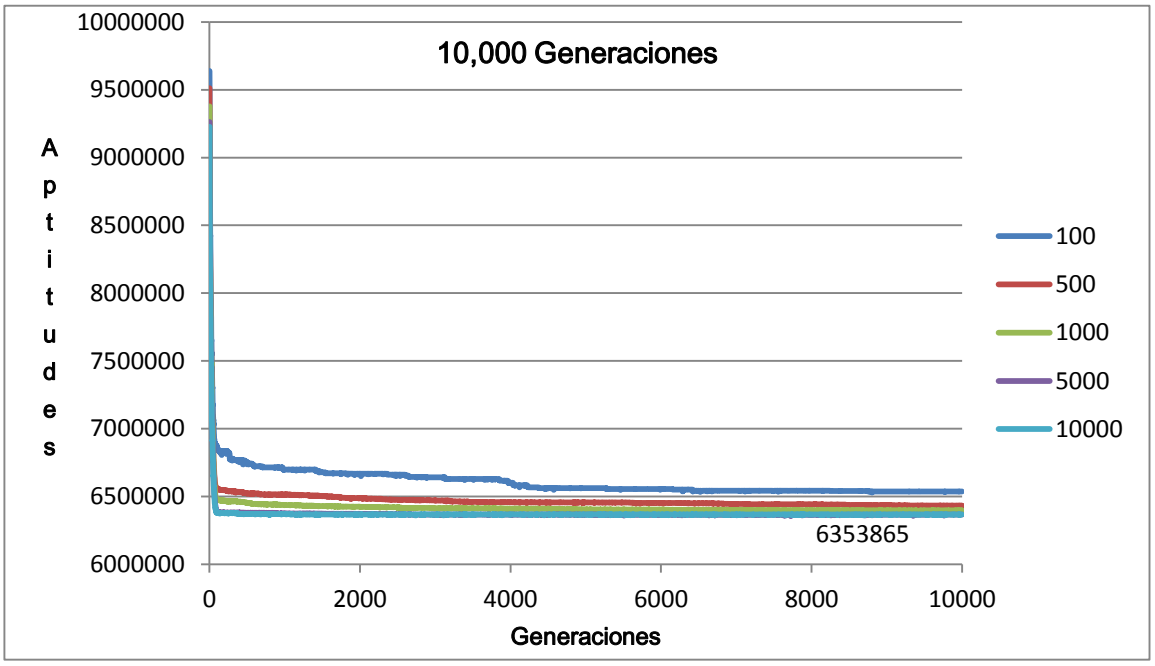


Figura 6.17 Diez mil Generaciones con Poblaciones de Diferente Tamaño

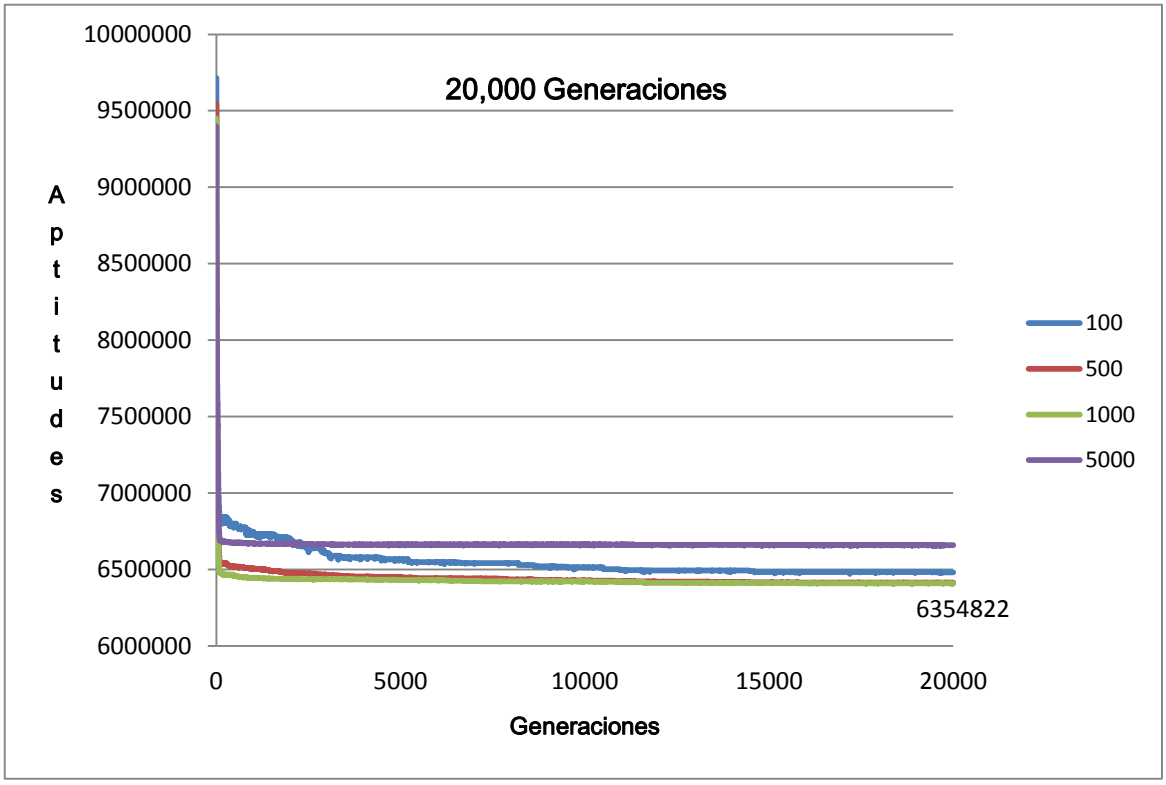


Figura 6.18 Veinte mil Generaciones con Poblaciones de Diferente Tamaño

Estos estudios experimentales, de análisis de sensibilidad para la instancia Hanoi, determinan que los valores más adecuados para los parámetros del algoritmo evolutivo se obtienen con 5,000 individuos, 5,000 generaciones, probabilidad de cruce de 80 y probabilidad de mutación de 20. Con estos valores, definidos para los parámetros del algoritmo evolutivo para la instancia Hanoi, el mejor valor de la función objetivo es de 6294141. Este valor se acerca a la mejor cota conocida, 6056370, con un error relativo de 3.9%.

Una vez sintonizados los parámetros de tamaño de población, probabilidad de cruce, probabilidad de mutación y número de generaciones, se procedió a realizar un último experimento para el parámetro del operador de mutación, parámetro que indica la cantidad de genes a mutar en un individuo. Para el operador de mutación se definen valores de 1, 3,10, 50 y 100. Estos valores indican el porcentaje de genes de un cromosoma que sufrirán alguna mutación. Con los resultados de este ajuste se observa que los mejores resultados del algoritmo son para valores del operador de mutación de 50 que indican que el cromosoma sufrirá cambios en la mitad de sus genes.

En diferentes trabajos en la literatura, el operador de mutación toma valores muy bajos, como por ejemplo 3-5, los cuales indican cambios en 3 a 5 por ciento del cromosoma. Algo importante que mencionar, es la diferencia en la implementación del algoritmo ya que la mayoría de los trabajos consultados y referenciados en el capítulo 2, mutan a todos los individuos descendientes, resultantes de la combinación de individuos. En este trabajo se ha implementado de forma distinta, considerando que de una población, de forma aleatoria algunos individuos de la población se combinan, mientras que otros sufren alguna mutación, pero no todos. En conclusión se puede decir que de acuerdo a los resultados obtenidos con el análisis de sensibilidad para la instancia de prueba Hanoi que se ha encontrado el mejor comportamiento del algoritmo con los valores de 5000 individuos, 5000 generaciones, probabilidad de cruce de 80, probabilidad de mutación de 20 y operador de mutación de 50.

6.2.2. Resultados de los experimentos para Instancia Hanoi

Además de los resultados mostrados en el análisis de sensibilidad previo para la instancia Hanoi, en este apartado se muestran resultados de los experimentos en cuanto al tiempo requerido para la ejecución del algoritmo. El tiempo empleado para el algoritmo evolutivo

utilizando diferentes tamaños de población y diferente número de generaciones puede verse en la gráfica de la Figura 6.19. La Figura muestra que el tiempo de ejecución del algoritmo evolutivo crece de manera exponencial al aumentar el número de generaciones y el tamaño de la población. Es esta la principal razón por la que las investigaciones experimentales trabajan con poblaciones 100-500 individuos consideradas como poblaciones pequeñas. Sin embargo, en este trabajo se hicieron pruebas con poblaciones de mayor tamaño para observar principalmente la eficiencia y eficacia del algoritmo evolutivo.

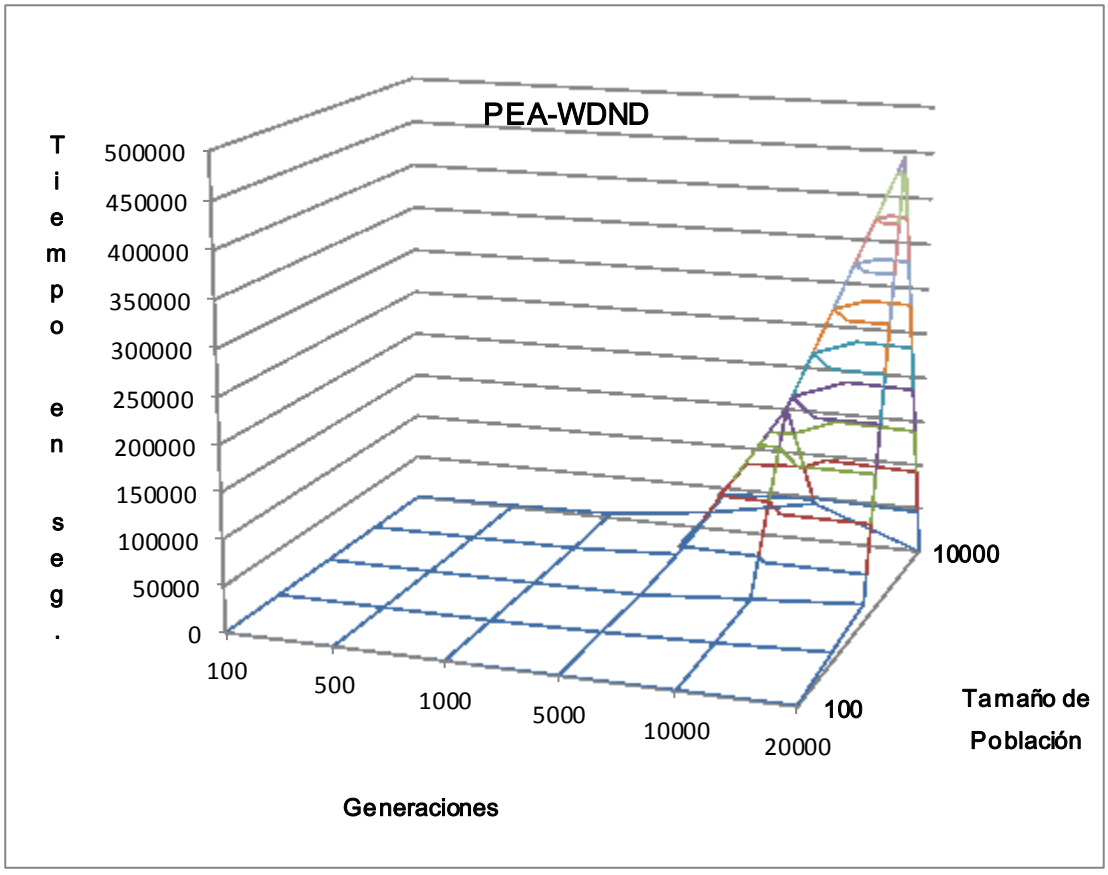


Figura 6.19 Tiempo de ejecución del Algoritmo Evolutivo Instancia Hanoi

6.2.3. Análisis de sensibilidad para la Instancia Balerma

El experimento de sintonización realizado para resolver la instancia Balerma consistió en la generación de múltiples pruebas, cada prueba incluyendo 30 ejecuciones del algoritmo SEA-WDND. Para esta instancia de prueba, se procede a la creación de la población inicial mediante una búsqueda local iterada.

El proceso de análisis de sensibilidad es similar para todas las instancias resueltas en este trabajo de tesis. Pero para instancias medianas y grandes los valores de los parámetros del algoritmo difieren considerablemente debido a la naturaleza del problema. Es decir, para las instancias pequeñas el algoritmo encuentra la mejor solución conocida en la literatura con tan solo realizar 20 generaciones del algoritmo, por lo que a lo sumo se han definido 1000 generaciones para los experimentos. No obstante, para resolver instancias medianas y grandes se requiere un número mucho mayor de generaciones como por ejemplo 20,000 o más. Otra diferencia importante en los estudios experimentales realizados para la sintonización del algoritmo, son los tamaños definidos de población. Por ejemplo, la instancia de prueba Alperovits o Hanoi, consideradas como pequeña y mediana respectivamente, pueden utilizar poblaciones de 5,000 o 10,000 individuos y con 20,000 generaciones y aun así el algoritmo termina su ejecución en tiempos de cómputo razonables que van desde minutos como mínimo a 3 horas como máximo. Para instancias de prueba grandes, como lo es Balerma, el tiempo y espacio requeridos para ejecutar el algoritmo con dichos parámetros incrementan considerablemente. Así, las poblaciones que se definen para la instancia Balerma son de 50,100, 500, 1,000, y 5000 individuos. Al número de generaciones para esta instancia se le asignan valores de 100, 1,000, 5000, 10,000 y 20,000 iteraciones. Una vez que se establecen los valores de prueba para los parámetros del algoritmo, se realizan 30 ejecuciones del algoritmo variando como primer parámetro el tamaño de la población para observar su comportamiento. Para este experimento se definieron valores iniciales para la probabilidad de cruce, la probabilidad de mutación y el número de generaciones, fijándose en 80, 20 y 1000 respectivamente. La creación de la población inicial para la instancia Balerma, consistió en un proceso de dos etapas. En la primera etapa se crea un individuo factible de forma determinista, mediante la asignación del mayor de los diámetros disponibles a todas las tuberías que forman la configuración de la red. Al hacer este

proceso se verifica que la configuración cumpla las restricciones del modelo hidráulico. Sin embargo, se observa que los individuos generados son infactibles en el módulo hidráulico, por tener presiones muy elevadas. Además de que, con estas configuraciones el costo de la red es elevado, de 21598384 €. Posteriormente, en una segunda etapa se aplica una serie de cambios a la configuración definida, con la finalidad de obtener diferentes individuos en la población que puedan combinarse y generar descendientes contribuyendo a la evolución de la población. Las soluciones infactibles en el módulo hidráulico se ajustan a factibles con la evolución de la población realizada por el algoritmo evolutivo. El primer experimento de análisis de sensibilidad para la instancia Balerma consistió en fijar los parámetros de Probabilidad de Cruce y Probabilidad de mutación con los mejores valores que se definieron como valores finales para las instancias pequeñas y medianas, siendo estos valores 80 y 20 respectivamente. Una vez fijos estos parámetros, los experimentos se realizaron con poblaciones de 50 y 100 individuos. Con estas poblaciones se define el operador de mutación (OpM) en 50 y 100, para observar la evolución de la población cuando de cada cromosoma a mutar se cambian todos los alelos en el caso de OpM=100. Así mismo observar la evolución cuando se cambian a lo mucho la mitad de los genes del cromosoma, OpM=50. Estos experimentos se realizaron para 100, 1000, 5000, 10,000 y 20,000 Generaciones del Algoritmo Evolutivo. La Figura 6.20 muestra las gráficas de las ejecuciones del algoritmo para poblaciones de 100 individuos. Todas las gráficas de la Figura muestran que el algoritmo evolutivo obtiene mejores resultados cuando el operador de mutación tiene el valor de 50, por lo que se descarta la opción de que un individuo pueda sufrir alteraciones en todos los genes del cromosoma. En esta figura se muestra la ejecución del algoritmo con 100 generaciones (Figura a), con 1000 generaciones (Figura b), con 5000 generaciones (Figura c), con 10,000 generaciones (Figura d) y con 20,000 generaciones (Figura e). En esta Figura se muestra la mejor ejecución del algoritmo, para el número de generaciones definido. Se observa que entre mayor el número de generaciones se mejora la aptitud de los individuos y por lo tanto menor es el costo del diseño de la red de distribución de agua. Se observa que la mejor cota encontrada con el algoritmo corresponde al valor de 2, 630,577, valor muy cercano a la mejor cota conocida 2, 302,423 [Reca, 2008]. Como conclusión de estos experimentos, es posible decir que la gran cantidad de estudios experimentales para las diferentes instancias de prueba permitieron determinar los valores de los parámetros del algoritmo SEA-WDND con los cuales éste tiene un mejor comportamiento,

para instancias de diferente tamaño consideradas como pequeñas, medianas y grandes. Gracias a este estudio se observó también el tiempo que requiere la ejecución del algoritmo cuando se modifican los valores de los parámetros, en especial se observó el incremento del tiempo cuando se incrementan los valores para el tamaño de la población y para el número de las ejecuciones del algoritmo.

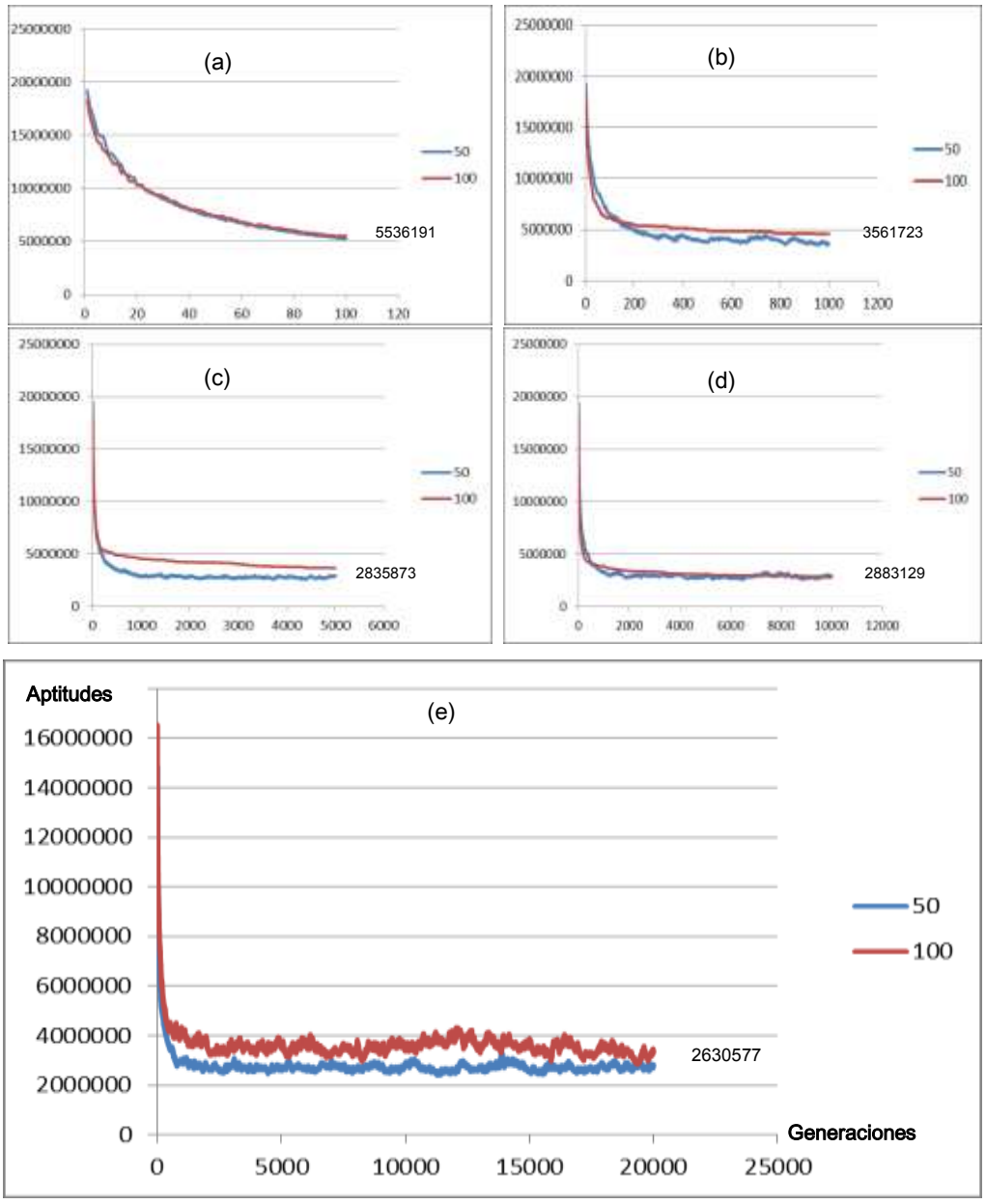


Figura 6.20 Operador de Mutación

Con los estudios experimentales se observó que la plataforma Grid utilizada para las pruebas del algoritmo está formada por 2 clúster homogéneos debido a las velocidades similares de ejecución del algoritmo. Con esta importante conclusión surgió la idea de aprovechar esta característica para la implementación del algoritmo paralelo PEA-WDND sin que requiera de rutinas sofisticadas para la sincronización de procesos.

6.3. Análisis de Sensibilidad del Algoritmo Evolutivo PEA-WDND

En este apartado se muestra el análisis de sensibilidad realizado para los valores paramétricos del Algoritmo Evolutivo en ambiente Grid, PEA-WDND. El análisis consiste en asignar valores diferentes a los parámetros que se refieren a la topología del algoritmo paralelo que se realiza con el modelo propio maestro-alumnos. En este estudio se pretende definir con cuáles valores de los parámetros el algoritmo tiene un mejor comportamiento. En concreto, los parámetros de estudio son: tamaño de los grupos, número de soluciones que se envían de los alumnos a los maestros y frecuencia de envío de soluciones (número de generaciones que transcurren entre cada entrega de las soluciones). Este estudio experimental, para el ajuste paramétrico del Algoritmo, se hace únicamente con la instancia de prueba Balerma, ya que es la que requiere de mayor cantidad de recursos computacionales. Las instancias medianas y pequeñas han sido resueltas de manera eficaz y eficiente con el algoritmo secuencial SEA-WDN desarrollado también en este trabajo doctoral. Las instancias medianas y pequeñas también han sido resueltas de manera eficaz y eficiente con el algoritmo paralelo observando mejora en tiempos de ejecución que disminuyen aun cuando el tamaño de las poblaciones incrementa.

El Algoritmo PEA-WDND tiene una amplia gama de posibilidades que pueden ser analizadas. La característica principal del algoritmo es que se implementa bajo el modelo maestro-alumnos y se ejecuta en ambiente grid. En este algoritmo cada alumno puede resolver el problema utilizando su propia estrategia, tal como ocurre en el ámbito escolar. Con esto el algoritmo hace una exploración exhaustiva tanto de los parámetros del algoritmo evolutivo, mostrados en el 6.2, como de diferentes métodos para los operadores del algoritmo evolutivo, descritos en el apartado 4.1.1. Así mismo el algoritmo PEA-WDND aprovecha el uso de los recursos existentes en la plataforma Grid Morelos, bajo un

modelo de paralelización que permite reducir tiempos de ejecución al trabajar con grupos, basándose en la idea del paradigma "*Divide y Vencerás*" muy conocido en cómputo distribuido. Cabe mencionar que el algoritmo PEA-WDND utiliza los mismos valores para las restricciones de presiones y velocidades definidas para las instancias en el apartado 6.2. Los experimentos de ajustes de parámetros realizados para resolver la instancia Balerma, con el algoritmo PEA-WDND, consisten en múltiples pruebas independientes. Las pruebas han sido ejecutadas en la Grid Morelos y antes de mostrar los resultados de las pruebas es conveniente describir detalladamente el estudio sistemático realizado para la ejecución del algoritmo. La Grid Morelos[6] cuenta con tres clúster de alto rendimiento: clúster CIICAp[7], el clúster Texcal[8] y el clúster Nopal[9].

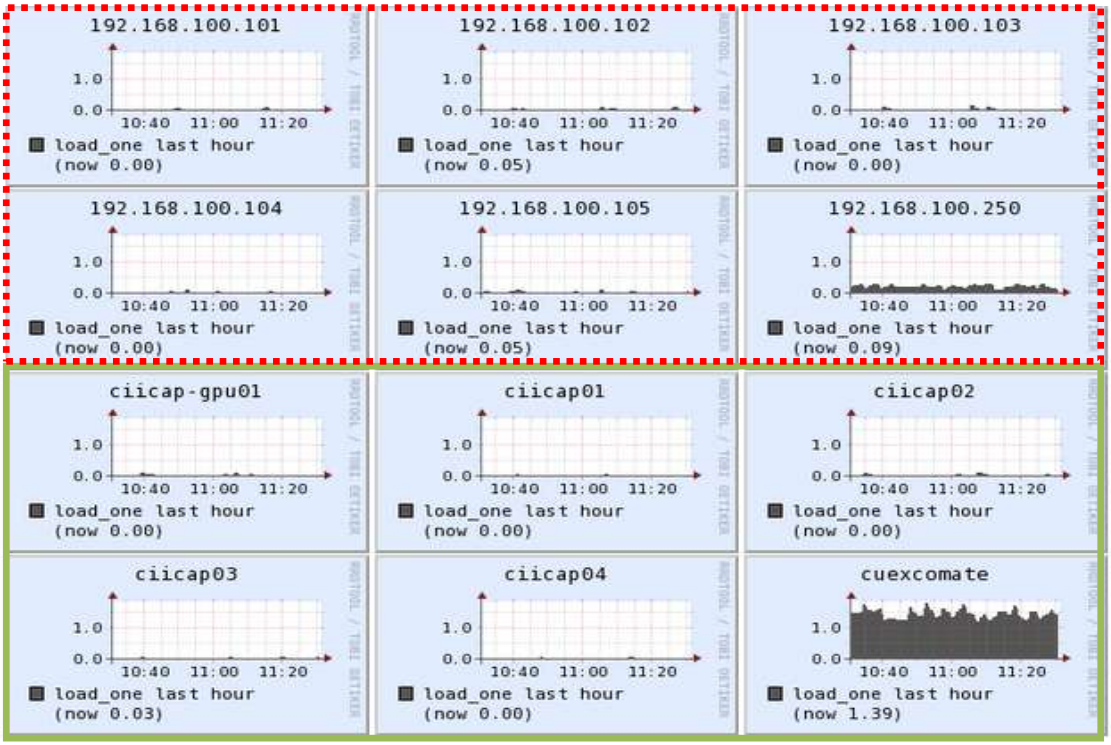


Figura 6.21 Estructura de la Grid de Pruebas

[6]http://www.gridmorelos.uaem.mx/ganglia/?m=load_one&r=hour&s=by%2520name&c=Cuexcomate&h=&sh=1&hc=3&z=small

[7] Centro de Investigación en Ingeniería y Ciencias Aplicadas.

[8] Universidad Politécnica del Estado de Morelos.

[9] Instituto Tecnológico de Veracruz.

La Figura 6.22 muestra los nodos de los clúster CIICAp (línea verde) y los nodos del clúster Texcal (línea roja). La unión de los clúster forma la Grid Morelos. La Grid Morelos puede considerarse como una plataforma heterogénea ya que cuenta con clúster de computadoras con diferentes características. Sin embargo, el clúster CIICAp y el clúster Texcal tienen características similares en cuanto al número de nodos y al número de núcleos, e incluso las características de velocidad de procesamiento y capacidad de almacenamiento, Tabla 6.10.

Tabla 6.10 Pruebas para 10,000 Individuos

	Clúster CIICAp		**Clúster Texcal**		
Nodo	**Nombre del nodo**	**Nodo**	**Nombre del nodo**	**Núcleos**	**Memoria**
1	***ciicap-gpu01***	7	***192.168.100.105***	6	37
2	***ciicap01***	8	***192.168.100.104***	12	24
3	***ciicap02***	9	***192.168.100.103***	12	24
4	***ciicap03***	10	***192.168.100.102***	12	24
5	***ciicap04***	11	***192.168.100.101***	12	24
6	***cuexcomate***	12	***192.168.100.250***	12	24

El número de nodos disponibles (Nodo) está definido en la Tabla 6.10. En esta tabla, se observa que en total son 12 los nodos disponibles en la Grid que se utilizan para las pruebas experimentales. En base al número de nodos existentes se ha determinado que el número máximo de grupos que se crean para la ejecución del algoritmo PEA-WDND será 12. Este valor es fijo pero puede escalarse con relativa facilidad para realizar mayor número de grupos en pruebas experimentales. El número de grupos está definido en el parámetro NGr y como valor mínimo para el funcionamiento adecuado del algoritmo es 1. Es decir, el Algoritmo PEA-WDND trabaja correctamente cuando se define mínimo un grupo o máximo 12 grupos. Otro parámetro importante del algoritmo evolutivo PEA-WDND es el número de elementos en el grupo, definido con el parámetro TamGr. A través de este parámetro se sabe que el número de alumnos en un grupo se define mediante NumAlum=TamGr-1, ya que el elemento restante es precisamente el maestro del grupo. Es importante decir que el tamaño del grupo debe ser mayor o igual a dos elementos, para poder apegarse al modelo definido en este trabajo: “maestro-alumnos”.

Una vez definidos los valores iniciales de los parámetros del algoritmo es posible realizar el análisis de sensibilidad.

6.2.3. Análisis de sensibilidad de PEA-WDND para la Instancia Balerma

Los estudios experimentales para el algoritmo evolutivo en ambiente Grid, PEA-WDND, se realizan considerando múltiples parámetros, definidos en la Tabla 6.11. En esta tabla se muestran parámetros propios del ambiente grid y también parámetros del algoritmo evolutivo. Algunos parámetros, como por ejemplo el tamaño del grupo de procesos en la grid, se definen con valores múltiplos de 6 debido a que el valor mínimo de núcleos que tienen los nodos (1 y 7) de los clúster CIICAp y Texcal respectivamente, es precisamente 6, véase Tabla 6.10.

Tabla 6.11 Parámetros para PEA-WDND

Ambiente Grid	Algoritmo Evolutivo
TamGr = {5,8,10,16}	*TamPob={100,200,1000}*
NumAlum= TamGr-1	*NG={10,000,20,000,30,000}*
Num_{sol}=*{1-TamPob/NumAlum}*	P_c=*{50-100}*
T_e = *{1-NG}*	P_m=*{10-50}*
	O_m=*{10-50}*

Otros parámetros, como el número de soluciones que los alumnos enviarán al maestro, están definidos mediante un rango. En el límite inferior del rango se indica que cada alumno del grupo debe enviar al menos una solución. En el límite superior del rango se indica que el número máximo de soluciones (individuos) que un alumno enviará está definido en función del tamaño de la población y del número de alumnos que haya, con la finalidad de que el maestro pueda tener como máximo el número de individuos necesarios para reemplazar la población completa. Así el valor de Num_{sol} es un valor aleatorio que se encuentra dentro del rango. Este valor lo define el maestro del grupo y lo da a conocer a los alumnos. El parámetro T_e es definido aleatoriamente también por el maestro y se hace en función del número de generaciones del algoritmo. Este valor indica la frecuencia de

comunicación entre alumnos y maestros. Así, si se elige el límite inferior del rango los alumnos se comunicarán con el maestro en cada generación, lo cual no es lo más recomendable en un algoritmo paralelo. Con el límite superior, los alumnos se comunicarían con el maestro únicamente al finalizar el número de generaciones del algoritmo. Como puede observarse se han definido límite inferior y límite superior para indicar un rango de valores. Sin embargo, el valor que el maestro establece para el parámetro T_e, en tiempo de ejecución, es un valor aleatorio que se encuentra dentro del rango definido. El primer estudio experimental en ambiente grid consistió en fijar los parámetros para el algoritmo evolutivo, PEA-WDND, y variar los parámetros relacionados con el modelo maestro alumnos que servirán para la ejecución del algoritmo en ambiente grid, Tabla 6.12.

Tabla 6.12 Parámetros para PEA-WDND

Ambiente Grid	Algoritmo Evolutivo
TamGr = {5,8,10,16}	*TamPob=200*
NumAlum= TamGr-1	*NG=20,000*
Num_{sol}= *TamPob/NumAlum*	P_c=0.8
T_e = *1000*	P_m=0.2
	O_m=0.03

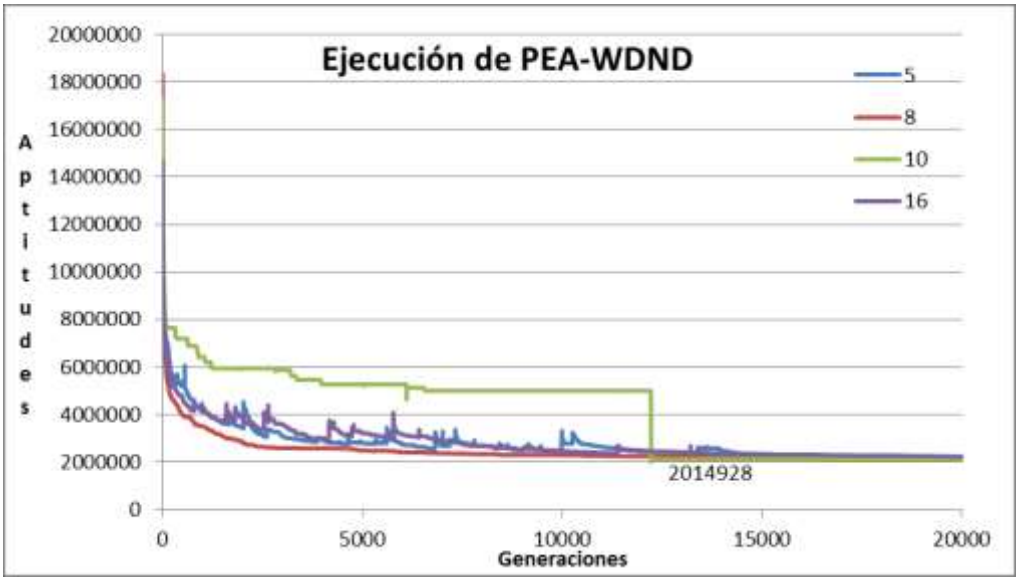


Figura 6.22 Grupos de diferente Tamaño

El estudio experimental realizado consistió en 30 pruebas experimentales, variando el tamaño del grupo. Es decir, para cada tamaño del grupo definido en TamGr, se realizaron 30 ejecuciones del algoritmo, por lo que en total se realizaron 120 ejecuciones del algoritmo para este estudio. Es importante decir que el tamaño de la población incrementaba con el número de procesos. Para el grupo de tamaño 5, en realidad se trabajaba con una población de 1000 Individuos. Así, para un grupo de tamaño 8, 10 y 16 el tamaño de la población fue de 1600, 2000 y 3200 respectivamente. Por otra parte, el número de soluciones que los alumnos envían al maestro está en función del tamaño de la población y del número de alumnos, de tal forma que el proceso maestro del grupo puede remplazar su población completa.

La Figura 6.23 muestra los resultados promedio de la ejecución del algoritmo, *PEA-WDND*, en ambiente grid utilizando diferente número de procesos, definidos todos ellos en la tabla 6.15. En esta gráfica se observa que el algoritmo tiene comportamiento satisfactorio utilizando cualquier número de procesos. Se observa también que el algoritmo es eficaz al encontrar valores promedio muy cercanos a la mejor cota conocida en la literatura para el problema de Diseño de Redes de Distribución de Agua [Reca, 2008]. En esta gráfica se observa que con el algoritmo PEA-WDND desarrollado en este trabajo de tesis se ha conseguido mejorar la mejor cota conocida para la red de distribución de agua Balerma. La mejor cota reportada en la literatura [Reca, 2008] es de 2, 630,577 Euros y el algoritmo PEA-WDND encuentra una solución para el diseño de la red de 2,014,928 Euros. En este primer estudio, se concluye que la mejor ejecución del algoritmo es cuando se trabaja con grupos de 10 individuos.

Un segundo estudio experimental consistió en definir nuevamente diferentes tamaños de grupos para el algoritmo PEA-WDND. En este estudio, el objetivo es determinar la eficiencia del algoritmo por lo que se procede a dividir el tamaño de la población entre el número de procesos disponibles teniendo así paralelización de datos, utilizando también el modelo de paralelización propuesto en este trabajo: Modelo Maestro-Alumnos. En este estudio, una población de 800 individuos se divide entre el número de nodos disponibles. Este estudio se realizó en los nodos: ciicap01, ciicap02, ciicap03 y ciicap04 porque tienen el mismo número de núcleos. Los parámetros del algoritmo, para este experimento, quedan definidos en la Tabla 6.13.

Tabla 6.13 Ejecución en la Grid para PEA-WDND

Ambiente Grid	Algoritmo Evolutivo
TamGr = 4	*TamPob=800*
NumAlum= TamGr-1	*NG=20,000*
Num_{sol}=*5*	P_c=*0.8*
T_e = *1000*	P_m=*0.2*
	O_m=*0.03*

Número de Prueba	Total de Procesos	Procesos por nodo	Población por Proceso
1	64	16	13
2	48	12	17
3	32	8	25
4	16	4	50
5	8	2	100
6	4	1	200
7	2	1	400
8	1	1	800

Con este estudio se muestra la eficiencia del algoritmo al reducir los tiempos de cómputo, mismos que están en función del número de procesos disponibles. Es importante notar que en este estudio el parámetro Num_{sol} que se refiere al número de soluciones a enviar cada vez que se establece la comunicación se ha definido con un valor fijo de 5, que indica el número de soluciones que enviarán los alumnos al maestro en cada 1000 generaciones.

La Figura 6.24 muestra costos promedio de 30 ejecuciones del algoritmo para la instancia de prueba Balerma, utilizando diferentes números de procesadores para la ejecución. La figura 6.25 está estrechamente relacionada con la figura 6.24 y muestra los tiempos promedio de 30 ejecuciones del algoritmo PEA-WDN. En el gráfico se puede apreciar que el tiempo disminuye considerablemente al incrementar el número de procesos en los que se ejecuta el algoritmo, e incluso se encuentra ligeramente por debajo del tiempo ideal para la ejecución de un algoritmo paralelo.

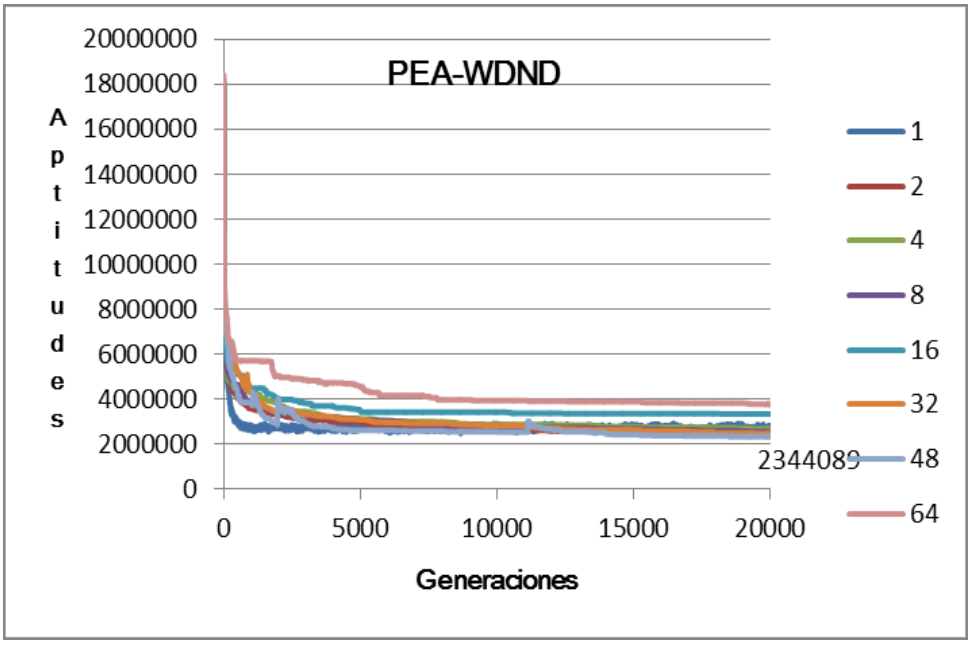


Figura 6.23 Ejecución de PEA-WDND para Instancia Balerma

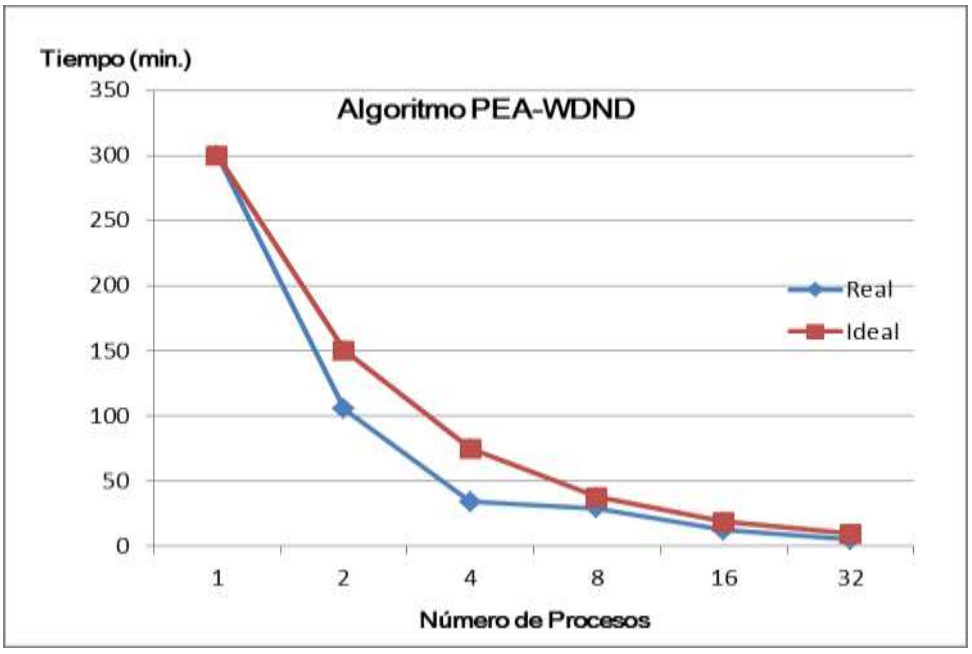


Figura 6.24 Tiempos de Ejecución PEA-WDND

La Figura 6.26 muestra los resultados de la mejor ejecución del algoritmo en ambiente grid utilizando diferente número de procesadores. Se observa con estos resultados que el algoritmo PEA-WDND obtiene excelentes resultados al mejorar los resultados presentados en la literatura con la mejor cota conocida [Reca, 2008].

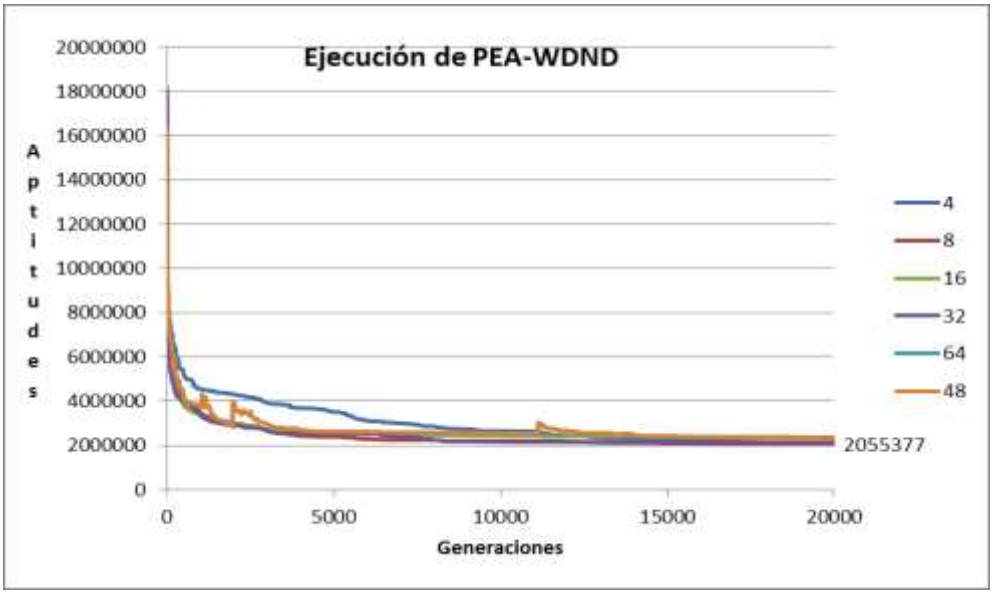


Figura 6.25 Ejecución PEA-WDND en Grupos de Distinto Tamaño

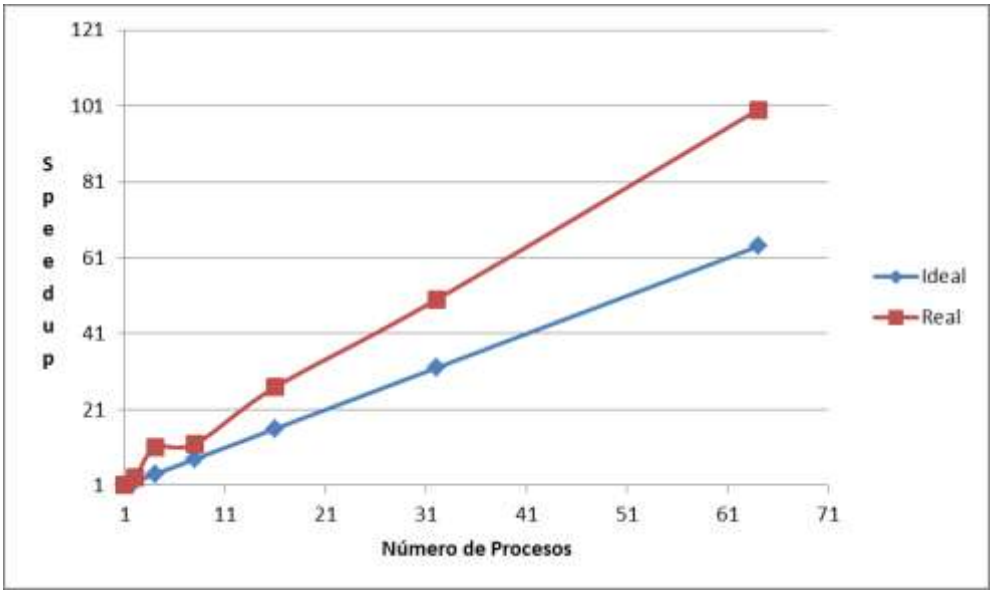


Figura 6.26 Ganancia de Velocidad de PEA-WDND

En la Figura 6.27 se muestra una gráfica de ganancia de velocidad que el algoritmo PEA-WDND obtiene en su ejecución con diferentes números de procesos. Se observa que la ganancia de velocidad es superlineal. Tal como se menciona en el apartado 4.5, en la práctica no es muy común encontrar la ganancia de velocidad ideal, si se considera que la ejecución de un algoritmo paralelo implica tener factores como latencia y tiempos de comunicaciones entre procesos. No obstante, el modelo de paralelización propuesto e implementado en este trabajo permitió que el algoritmo evolutivo, PEA-WDND, desarrollado lograra una ganancia superlineal. El modelo maestros-alumnos fue cuidadosamente pensado e implementado de tal forma que los factores de latencia y tiempos de comunicaciones no afectaran el desempeño del algoritmo.

La implementación del algoritmo PEA-WDND con el modelo maestro-alumnos obtiene excelentes resultados gracias a que las comunicaciones son relativamente esporádicas y a que la información viaja serializada. Es decir, gran parte del éxito del algoritmo evolutivo en ambiente grid se debe al compromiso logrado entre envío de información entre procesos y a al número de paquetes enviados. Con este modelo implementado para el algoritmo PEA-WDND se logra la explotación de recursos tales como memoria principal y número de procesadores obteniendo valores de ganancia de velocidad por encima de los valores lineales. También, es importante mencionar que el éxito del algoritmo se debe en parte al diseño cuidadoso del algoritmo secuencial, mismo que fue implementado pensando en optimizar todas y cada una de sus funciones. Por ejemplo, la función de ordenación, el método de búsquedas de registros. Estas funciones han sido posibles gracias a las estructuras del algoritmo definidas y utilizadas. Se puede concluir que en este trabajo se ha hecho uso adecuado del procesamiento paralelo logrando que el algoritmo SEA-WDND llevado a entorno paralelo PEA-WDND en base a la prueba de que este último obtiene una ganancia de velocidad de superlineal.

6.4. Comparación de Resultados

La comparación de la calidad del algoritmo evolutivo desarrollado en este trabajo se determina con base en la comparación de las soluciones que obtiene este algoritmo, con los resultados de trabajos de otros investigadores. La Tabla 6.14 muestra las mejores soluciones obtenidas por algoritmos de diferentes autores. Es importante decir que en

estos estudios, el algoritmo evolutivo considera todas las restricciones del modelo matemático de satisfacción de restricciones. Se observó en los experimentos que cuando se consideran todas las restricciones de dicho modelo, los resultados mínimos obtenidos son ligeramente mayores a cuando no se considera la restricción de velocidades. Cabe mencionar que la mayoría de los autores no consideran velocidades en la red puesto que la restricción está estrechamente relacionada con la restricción de presiones. Solo que al considerarla también, se añade complejidad al problema. Así con las restricciones de velocidades el costo mínimo encontrado es de 420,000 unidades monetarias. Sin considerar velocidades el costo mínimo fue de 419, 000 unidades, el cual se considera que es la mejor solución al problema de Diseño de Redes de Distribución de Agua para la instancia Alperovits. Otro de los aspectos que se mencionan en la literatura, que ayudan a tener una red de menor costo es configurar a Epanet para que utilice diferentes fórmulas para el cálculo de pérdidas de carga, lo cual quedó fuera del alcance de este trabajo por considerarse un tema más completo del área de hidráulica y este trabajo se enfoca principalmente en el área de optimización.

Tabla 6.14 Resultados de diferentes autores para Instancia Alperovits

Fecha	Autores	Método	Costo
[1997, Savic]	Savic y Walters	AG	419000
[1998, Abebe]	Abebe et al.	AG	424000
[1999, Montesinos]	Montesinos et al.	AG	456000
[2003, Matias]	Matías	AG	419000
[2008, Reca]	Reca et al.	AG	419000
SEA-WDND	**Avila et al.**	**AE**	**419000**

La Tabla 6.15 muestra los parámetros de los algoritmos, de diferentes autores, con los que se obtiene la mejor solución, para la instancia de prueba Alperovits. Se observa en ella que al igual que otros autores el Algoritmo Evolutivo desarrollado en este trabajo encuentra la solución de 419, 000 la cual se cree que es la solución óptima para el problema de diseño de la red Alperovits. La Tabla 6.16 muestra los parámetros del algoritmo para la instancia de prueba Hanoi. La Tabla 6.17 muestra los resultados de la ejecución del algoritmo cuando soluciona la instancia Hanoi y se observa que el algoritmo es eficaz al resolver el problema acercándose a la mejor cota conocida en la literatura.

Tabla 6.15 Ajuste de parámetros para Instancia Alperovits

FECHA	AUTORES	GENOTIPO	PARÁMETROS	COSTO OBTENIDO
1997	Savic y Walters	Binaria (Código Gray)	TP=50 NG=50-500 Pc=1.0 Pm=0.03	419000
1999	Montesinos et al.	Binaria	TP=300 NG=300 Pc=1.0 Pm=0.03(two-loop)	456000
1999	Matías	Binaria (Código Gray)	TP=100-1000 NG=100-1000 Pc= Pm=	419000
2006	Reca et al.	Enteros	TP=500 NG=300 P_c=0.9 P_m=0.05 r_{cross}=0.5	419000
2010	**SEA-WDND**	Flotantes	TP=800 NG=100 P_c=0.9 P_m=0.1 O_m=0.5	419000

Tabla 6.16 Ajuste de parámetros para Instancia Hanoi

FECHA	AUTORES	GENOTIPO	PARÁMETROS	COSTO OBTENIDO
1997	Savic y Walters	Binaria (Código Gray)	TP=50 NG=50-500 Pc=1.0 Pm=0.03	6073000
2006	Reca et al.	Enteros	TP=500 NG=300 P_c=0.9 P_m=0.05 r_{cross}=0.5	6081127
2010	**SEA-WDND**	Flotantes	TP=800 NG=100 P_c=0.8 P_m=0.2 O_m=0.03	**6294141**

Tabla 6.17 Resultados de diferentes investigadores para la instancia Hanoi

Referencia	Autores	Método	Costo
[Savic,1997]	Savic and Walters	AG	6073000
[Abebe,1998a]	Abebe and Solomatine	AG	7000600
[Vairavamoorthy,2000]	Vairavamoorthy and Ali	AG	6056370
[Eusuff,2003]	Eusuff and Lansey	AG	6073000
[Matias,2003]	Matías et al.	AG	6093498
[Reca, 2006]	Reca et al.	AG	6081127
[Reca, 2008]	Reca et al.	AG	6081127
SEA-WDND	**Avila et al.**	**AE**	**6294141**

6.5. Resultados de la Experimentación

Los experimentos de sintonización, para el algoritmo evolutivo, se realizaron en instancias de prueba pequeñas medianas y grandes. Con los resultados obtenidos se muestra la convergencia del algoritmo encontrando soluciones de calidad para el diseño de diferentes redes de distribución de agua.

El algoritmo evolutivo PEA-WDND puede aplicarse a otros problemas de diseño o mantenimiento de redes existentes, ya que el algoritmo es capaz de detectar a los usuarios de la red de distribución de agua no tienen un servicio acorde a sus necesidades y con ello cambiar algunas tuberías para que, con el nuevo diseño de la red, se ofrezca a todos los usuarios de la red un servicio que satisface sus necesidades hídricas.

En este trabajo se utilizó una instancia de prueba real, correspondiente a la red de distribución de agua Balerma. Los resultados obtenidos con el algoritmo evolutivo secuencial, PEA-WDND, son resultados de calidad para todas las instancias probadas, ya que se acercan o incluso mejoran a las mejores cotas conocidas en la literatura.

6.6. Problemas resueltos Algoritmo Evolutivo PEA-WDND

En las primeras versiones se tenía el problema de que al lanzar el algoritmo para ser ejecución en paralelo los resultados de crear la población eran idénticos. Podría decirse que era un solo proceso quien creaba la población y las demás eran replicas con la duda de que cada proceso creaba su propio archivo con diferente nombre pero el contenido del archivo que era la población creada era realmente el mismo. Ante esto se decidió crear diferente semilla utilizada para la generación de números aleatorios antes de iniciar la creación de la población. Con esta estrategia se resuelve el problema, verificándolo cuando cada proceso su población inicial distinta. Incluso también resultó necesaria l se modificación de la semilla utilizada para la generación de números aleatorios cuando se crean los grupos. Esto con la finalidad de que en cada ejecución del algoritmo los maestros sean distintos, si es que así se desea.

Migración del Algoritmo trabajando en el clúster a trabajar en la grid. Existieron diferentes complicaciones, desde la plataforma que se utilizaba en el clúster con distintas características a la grid. En el clúster la plataforma es de 32 bits mientras que en el clúster la plataforma es de 64 bits. Fue necesario buscar Epanet para 64 bits y crear nuevamente el enlace del módulo de optimización con el módulo Epanet. Además de que al trabajar en la grid ocurrían problemas diversos por intentar acceder a la biblioteca Epanet al mismo tiempo. Este problema se resolvió haciendo que los procesos accedieran una sola vez a la configuración de la red y cargando dicha configuración en memoria para evitar la implementación de técnicas de sincronización desde el acceso a la información de la red de distribución de agua a resolver. Otro problema identificado y resuelto fue el uso de las variables en MPI. Es importante decir que en MPI las variables deben ser globales, de otra manera el algoritmo no funcionará de manera correcta porque las variables definidas como locales no se actualizan con el envío y recepción de mensajes.

Capítulo 7. Conclusiones y Trabajos Futuros

En este capítulo, se presentan las conclusiones obtenidas al finalizar este trabajo de tesis doctoral. También, se presentan las contribuciones y aportaciones del trabajo desarrollado. Finalmente, se describen los trabajos futuros, que podrían dar continuidad a este trabajo de investigación.

El algoritmo evolutivo desarrollado en este trabajo de tesis ha sido el resultado de una gran cantidad de experimentos computacionales. Todo este trabajo se realizó con la finalidad de solucionar un problema de optimización complejo, como lo es el problema de Diseño de Redes de Distribución de Agua. El algoritmo desarrollado ha sido utilizado para resolver instancias de prueba teóricas y una instancia de prueba real, llamada Balerma que contiene información procedente de la red de distribución de agua del municipio de Almería, España. Los requisitos que deben satisfacerse, para dar solución al problema de Diseño de Redes de Distribución de Agua, han sido definidos mediante la formulación de dos modelos independientes que representan, por un lado el diseño de la red y por otro el comportamiento que debe tener la red para ofrecer a los usuarios un servicio adecuado acorde con sus necesidades. Con las restricciones definidas para el diseño y operación de la red, encontrar la solución óptima para resolver el problema del diseño de la red de distribución de agua es realmente una tarea de gran complejidad que implica el uso de gran cantidad de recursos computacionales.

El problema de Diseño de Redes de Distribución de Agua consiste en encontrar la configuración de la red de costo mínimo. A la vez que también se garantice que la operación de la red es adecuada y puede satisfacer las necesidades hídricas de los usuarios. Para resolver el problema, se han tomado en cuenta, de manera fidedigna, cada uno de las restricciones del modelo matemático definido para el problema.

La solución al problema de Diseño de Redes de Distribución de Agua se obtiene mediante la técnica de Algoritmos Evolutivos en ambiente Grid. Ésta ha sido la estrategia que mejores resultados ofrece para el problema abordado, ya que ofrece resultados de calidad en tiempos razonables, lo cual es el objetivo principal del área de optimización dentro de la cual se encuentra el presente trabajo.

Esta tesis doctoral, ha sido el resultado de un trabajo exhaustivo de investigación en el que se han cubierto temas relacionados con el problema de redes de distribución de agua, temas del funcionamiento y operación de los algoritmos evolutivos y temas de algoritmos paralelos abriendo así nuevos horizontes de investigación en cada uno de los temas. A pesar de que los algoritmos evolutivos han sido utilizados frecuentemente para solucionar problemas de optimización, no se encontró en la literatura el empleo de la técnica desarrollada en este trabajo, para dar solución al problema de Diseño de Redes de Distribución de Agua. La importancia del desarrollo del algoritmo evolutivo al ser implementado en ambiente grid es que consigue explotar al máximo los recursos computacionales a los cuales se tiene acceso mediante un modelo híbrido de paralelización propuesto en este trabajo, llamado modelo maestro-alumnos. Con el uso del modelo maestro-alumnos el algoritmo PEA-WDND obtiene resultados de calidad en tiempos razonables para problema de Diseño de Redes de Distribución de Agua.

7.1. Conclusiones

El Algoritmo Evolutivo desarrollado en este trabajo de tesis se enfoca en el estudio de los componentes del Algoritmo para observar cómo se afecta su comportamiento. En este trabajo se han realizado experimentos para instancias de prueba consideradas como pequeñas, medianas y grandes. La instancia Alperovits, considerada como pequeña ha sido parte primordial de este trabajo, ya que con ésta su posible diseñar y sobre todo validar los resultados del Algoritmo Evolutivo. Fue posible comprobar que las soluciones dadas por el algoritmo realmente sean soluciones válidas gracias a que la comprobación de los resultados es relativamente fácil. Después de un análisis de los resultados se han determinado los parámetros del algoritmo que permiten encontrar soluciones de calidad para el problema de Diseño de Redes de Distribución de Agua. También puede concluirse, gracias a los estudios experimentales realizados, que la calidad de los

individuos de la población inicial en el algoritmo evolutivo influye positivamente en el éxito de encontrar excelentes soluciones para el problema de Diseño de Redes de Distribución de Agua abordado. En este trabajo de tesis se realizaron miles de experimentos tanto para el diseño del Algoritmo como para las pruebas del mismo. También se hicieron experimentos de validación de cada una de las funciones del Algoritmo Evolutivo utilizando el paradigma divide y vencerás. Con las pruebas de validación es posible decir que el algoritmo tiene un funcionamiento correcto para la solución de problemas de diseño o re-Diseño de Redes de Distribución de Agua. Con los resultados obtenidos de las pruebas experimentales del algoritmo PEA-WDND se justifica el uso de la grid. Se puede decir que esta importante herramienta ha impactado de forma positiva para la solución del problema de Diseño de Redes de Distribución de Agua.

El éxito del algoritmo se debe en gran parte a la metodología de solución y al modelo de paralelización propuesto e implementado en este trabajo de tesis doctoral. Así el algoritmo evolutivo en ambiente grid trabaja de manera eficiente encontrando soluciones de calidad en un tiempo de cómputo limitado para el problema de Diseño de Redes de Distribución de Agua, cumpliendo así el objetivo principal de este trabajo de tesis. En este trabajo se ha hecho uso adecuado del procesamiento paralelo logrando que el algoritmo SEA-WDND llevado a un entorno paralelo. Todo lo anterior fundamentado en los resultados de calidad obtenidos por el algoritmo PEA-WDND, obteniendo además una ganancia de velocidad de superlineal, Figura 6.27.

Los objetivos planteados al inicio del trabajo doctoral han sido cumplidos obteniendo:

1. Formulación de un modelo matemático que represente el diseño de una red de distribución de agua en el que el objetivo es brindar a los usuarios de la red un buen servicio (tanto en la operación de la red como en el servicio de los usuarios), con presiones mínimas requeridas para satisfacer sus demandas hídricas (Capítulo 3).
2. Implementación de un algoritmo evolutivo en ambiente Grid para solucionar el modelo matemático y obtener soluciones de calidad para el problema de Diseño de Redes de Distribución de Agua (capítulo 3 y 4).

3. Estudios experimentales para probar que el método de solución propuesto es eficaz y eficiente para el problema de Diseño de Redes de Distribución de Agua (Capítulo 6).
4. Comparación de la eficiencia y la eficacia del algoritmo evolutivo paralelo con versiones propias de algoritmos secuenciales (Capítulo 6), concluyendo que el algoritmo paralelo PEA-WDND es igual de eficiente pero más eficaz que el algoritmo secuencial SEA-WDND.
5. Comparación la eficiencia y la eficacia del algoritmo evolutivo paralelo con los resultados obtenidos por algoritmos de diferentes investigadores, reportados en la literatura (Capítulo 6) , concluyendo que el algoritmo paralelo PEA-WDND es igual de eficaz para algunas instancias(Alperovits) pero más eficaz para otras(Balerma).

Finalmente, se ha logrado la difusión de los resultados con una serie de publicaciones en revistas y en congresos, Apéndice IV.

7.2. Trabajos Futuros

Algunos puntos importantes que podrían dar continuidad a este trabajo de tesis son los que a continuación se enlistan:

- En cuanto al problema de Diseño de Redes de Distribución de Agua serían trabajar con nuevos planteamientos del problema para el diseño como por ejemplo en lugar de tener que distribuir el agua en una red de gran tamaño, evaluar el caso de tener redes más pequeñas para observar si se mejora el servicio. Otro punto importante sería optimizar el costo de la distribución de agua en la red cuando se utiliza la técnica de distribución por bombeo.
- En cuanto al algoritmo evolutivo, un trabajo futuro sería trabajar también con poblaciones infactibles para ver su evolución. También podría continuarse con la definición de parámetros variando valores, tanto para el Algoritmo Evolutivo como para su ejecución con el modelo maestro-alumnos.
- En cuanto al modelo de paralelización pueden formarse más grupos, ya que el algoritmo está limitado por ahora a máximo 12 grupos. También puede implementarse comunicación entre maestros, porque por ahora los maestros solo

se comunican con los alumnos. También podría implementarse en el modelo estrategias de distribución de carga dinámica de tal forma que se asigne nuevamente trabajo a los alumnos que finalicen. Otro trabajo futuro del modelo sería la implementación de nuevas técnicas de sincronización para poder diseñar algoritmos en ambientes heterogéneos.

- En cuanto a las instancias de prueba pueden realizarse pruebas para el diseño o re-diseño de nuevas instancias proporcionando archivos que requiere el Algoritmo para obtener información de la red. Pueden hacerse pruebas con diseños de redes reales, de pequeño, mediano e incluso de gran tamaño como lo es la red Balerma.

Apéndice I. Instancias de Prueba

En este apartado se presentan datos de las instancias de prueba utilizadas en este trabajo doctoral.

Tabla 1. Diámetros Comerciales y Costos para red Alperovits

Diámetros (pulgadas)	Diámetros (milímetros)	Costo (unidades)
1	25.4	2
2	50.8	5
3	76.2	8
4	101.6	11
6	152.4	16
8	203.2	23
10	254.0	32
12	304.8	50
14	355.6	60
16	406.4	90
18	457.2	130
20	508.8	170
22	558.8	300
24	609.6	550

Tabla 2. Diámetros Comerciales y Costos para red Hanoi

Diámetros (pulgadas)	Diámetros (milímetros)	Costo (unidades)
12	304.8	45.73
16	406.4	70.4
20	508.8	99.38
24	609.6	129.33
30	762	180.74
40	1016	278.28

Tabla 3.8. Diámetros Comerciales y Costos para red Balerma

Diámetros (milímetros)	Costo (unidades)
125	7.22
140	9.10
160	11.92
180	14.84
200	18.38
250	28.60
315	45.39
400	76.32
500	124.64
630	215.85

Apéndice II. Código Fuente del Algoritmo Evolutivo

En este apartado se presentan fragmentos de código del Algoritmo Evolutivo desarrollado en este trabajo de tesis. Se presentan las funciones más representativas del Algoritmo Evolutivo SEA-WDND, que corresponde al algoritmo evolutivo secuencial para el problema de Diseño de Redes de Distribución de Agua.

En este apartado, también se presentan fragmentos de código de funciones propias del algoritmo evolutivo PEA-WDND que corresponde al algoritmo evolutivo paralelo para el mismo problema abordado en esta tesis. También se presentan funciones para la comunicación de los procesos, en ambiente Grid.

Considerando que ambos algoritmos presentan funciones comunes, en este apartado se presentan dichas funciones comunes.
Finalmente, en este apartado, se presenta también un script desarrollado para el análisis de los resultados del algoritmo evolutivo. Este script se llama SQL-EAWDND.

```
/*-------------------------------------------------------------------------
Inicio de la funcion principal
-------------------------------------------------------------------------*/
int main (int argc, char** argv)
{

    int I=0,J=0,K=0,k=0,band=0,NumGrupos,rol=0,Grupo[NumIndivGrupo];
    int NumAlum=NumIndivGrupo-1;
    int AlumnosGrupo[NumAlum],j=0;
    int salida=0;
    FILE *fp;
    char nombreArch[20];
    char nombresArch[20];
    AgendaTE=TiempoEntr;
    sprintf(nombreArch,"ArchResultados%d%s",IdProceso,".txt");
    //Se inicializa el entorno de ejecucion MPI
     mpi_error_code = MPI_Init(&argc, &argv);
     RevisaErroresMPI(mpi_error_code, "MPI_Init", __LINE__);
    (void) time(&tiempoInicio);
    //Se almacena el identificador del proceso en MPI_COMM_WORLD
     mpi_error_code=MPI_Comm_rank(MPI_COMM_WORLD,&IdProceso);
    RevisaErroresMPI(mpi_error_code, "MPI_Comm_rank", __LINE__);
    //Se almacena el numero de procesos
     mpi_error_code=MPI_Comm_size(MPI_COMM_WORLD,&NumProcesadores);
    RevisaErroresMPI(mpi_error_code, "MPI_Comm_Size", __LINE__);
    stacksize_();
    //crea grupos con alumnos y profesor
    NumGrupos=NumProcesadores/NumIndivGrupo;
    while(I < NumGrupos && !salida)
    {
        LimiteInferior = I*(NumIndivGrupo);
        LimiteSuperior = (I+1)*(NumIndivGrupo);
        if(IdProceso  >= LimiteInferior  && IdProceso < LimiteSuperior)
        {
            for (J=LimiteInferior; J<LimiteSuperior; J++)
            {
                 Grupo[k]=J;
                 k++;
             }
             grupo=I+1;
             salida=1;
         }
     I++;
     }
     switch(grupo)
     {
         case 1:NuevoCom=grupo1;
             new_group=g1;
             break;
         case 2:NuevoCom=grupo2;
             new_group=g2;
             break;
         case 3:NuevoCom=grupo3;
             new_group=g3;
             break;
         case 4:NuevoCom=grupo4;
             new_group=g4;
             break;
     }
```

```
MPI_Comm_group(MPI_COMM_WORLD, &orig_group);
MPI_Group_incl(orig_group,NumIndivGrupo,Grupo, &new_group);
MPI_Comm_create(MPI_COMM_WORLD, new_group, &NuevoCom);
MPI_Group_rank(new_group, &IdSubProceso);
if(IdProceso==0)
{
    for (I=0; I<NumGrupos; I++)
    {
        LimiteInferior = I*(NumIndivGrupo);
        LimiteSuperior = (I+1)*(NumIndivGrupo);
        ListaMaestros[I]=GeneraNumAleatorio(LimiteInferior,LimiteSuperior);
    }

}
MPI_Bcast (&ListaMaestros,NumGrupos, MPI_INT,0, MPI_COMM_WORLD);
J=0;
while (J < NumGrupos && band !=1)
{
    if(IdProceso==ListaMaestros[J])
    {
        rol=1;
        band=1;
    }
    J++;
}

if(rol == 1)
    RealizaActividadesMaestro();
else
    RealizaActividadesAlumno();
MPI_Finalize();
return EXIT_SUCCESS;
}
```

```
/*--------------------------------------------------------------------------
 Los maestros establecen las reglas a seguir y los problemas a resolver
--------------------------------------------------------------------------*/
void RealizaActividadesMaestro()
{

    int NumSoluciones=0;
    int NumAlum=NumIndivGrupo-1;
    int AlumnosGrupo[4], Grupo[5];
    int I=0,salida=0,J,j=0,k=0,NumGrupos;
    grupo=0;
    srand((unsigned)time(NULL)+IdProceso);//semilla
    los maestros conocen su grupo y envian a sus alumnos información importante
    if(corridas == 0)
        DefineParametros();
    NumSoluciones=GeneraNumAleatorio(1,TamPoblacion/NumAlum);
    NumSolRecibir=NumSoluciones;
    Actividades[0]=NumSolRecibir;
    tiempos de entrega
    TiempoEntr=GeneraNumAleatorio(1,NumGeneraciones);
    Actividades[1]=TiempoEntr;
    NumGrupos=NumProcesadores/NumIndivGrupo;
    gethostname(hostname, 90);
    se define grupo
    while(I < NumIndivGrupo && !salida){
        LimiteInferior = I*(NumIndivGrupo);
        LimiteSuperior = (I+1)*(NumIndivGrupo);
        if(IdProceso  >= LimiteInferior  && IdProceso < LimiteSuperior){
            for (J=LimiteInferior; J<LimiteSuperior; J++)
                if(IdProceso != J){
                    AlumnosGrupo[j]=J;
                     j++;
                 }
             salida=1;
             grupo=I+1;
        }
        I++;
    }
    Avisa a los alumnos las politicas de evaluación
    for (I=0; I<NumIndivGrupo; I++) {
        if(IdSubProceso != I){
            if(MPI_Send(&Actividades, 2, MPI_INT,I,etiqueta,NuevoCom) == MPI_SUCCESS){
                printf("\n\nDatos Enviados con Exito");
            }
            else{
                printf("\n\nError de Envio");
                exit(1);
            }

        }
    }
    while(corridas  < 30){
        EvaluaResultados();
        corridas++;
    }
```

```
/*--------------------------------------------------------------------------
 Los alumnos realizan las actividades para resolver el problema
--------------------------------------------------------------------------*/
void RealizaActividadesAlumno()
{
    int n=0,J=0,I=0,salida=0;
    int id=0,NumGrupos;
    FILE *ffp;
    char nombreArch[20];
    LimiteInferior=0;
    LimiteSuperior=0;
    NumGrupos=NumProcesadores/NumIndivGrupo;
    grupo=0;
    if(corridas == 0)
        DefineParametros();

    while(I < NumIndivGrupo && !salida)
    {
        LimiteInferior = I*(NumIndivGrupo);
        LimiteSuperior = (I+1)*(NumIndivGrupo);
        if(IdProceso  >= LimiteInferior  && IdProceso < LimiteSuperior)
        {
            salida=1;
            grupo=I+1;
        }
        I++;
    }
    //identifican al profesor
    J=0;
    while(J < NumGrupos && !encontrado)
    {
        if(ListaMaestros[J] >= LimiteInferior  && ListaMaestros[J] < LimiteSuperior)
        {
            for (I=LimiteInferior; I<LimiteSuperior; I++)
            {
                if(ListaMaestros[J]==I)
                    profe=id;
                else
                    id++;
            }
            encontrado=1;
        }
    J++;
    }
    if(MPI_Recv(&Actividades,2, MPI_INT,profe,etiqueta,NuevoCom, &estado)!= MPI_SUCCESS
    {
            printf("\n\nError de Recepcion en el esclavo");
            exit(1);
    }
    else
    {
        NumSolEntregar=Actividades[0];
        TiempoEntr=Actividades[1];
    }
    while(corridas  < 30)
    {

        EvolucionaPoblacion();
        corridas++;
    }
```

```
/*------------------------------------------------------------------------
Cada proceso define la estrategia que aplicará para solucionar problema
------------------------------------------------------------------------*/
void DefineEstrategiaSolucion()
{
    //define cruce y mutacion como eventos 1)dependientes o 2)independientes
    int operador[2]={1,2};
    int Estrategia[5];
   //int Operadores, ProbCr,PuntoCr,ProbMuta,TipoSeleccion,TipoCr;

    //parametros del algoritmo sobre que porcentaje de la población cruzar o mutar
    int PrCruce[10]={10,20,30,40,50,60,70,80,90,100};
    int PrMuta[10]={10,20,30,40,50,60,70,80,90,100};
    // puede ser monopunto o multipunto
    int TipoCruza[2]={1,2};
    //el punto de cruce puede elegirse aleatoriamente o deterministicamente
    int PuntoCruce[2]={1,2};
    int Seleccion[4]={1,2,3,4};
    int PPrCr,PPrMu,PTiCr,PPuCr,PSele;

    POpera=GeneraNumAleatorio(0,1);
    Operadores=operador[POpera];//cruce y muta eventos dependientes o independientes
    PPrCr=GeneraNumAleatorio(0,9);
    ProbCr=PrCruce[PPrCr];
    if(POpera==1)
    {
        PPrMu=100-PrCruce[PPrCr];
        ProbMuta=PPrMu;
    }
    else
    {
        PPrMu=GeneraNumAleatorio(0,9);
        ProbMuta=PrMuta[PPrMu];
    }
    PTiCr=GeneraNumAleatorio(0,1);
    TipoCr=TipoCruza[PTiCr];//cruce monopunto o multipunto
    PPuCr=GeneraNumAleatorio(0,1);
    PuntoCr=PuntoCruce[PPuCr];//punto de cruce
    PSele=GeneraNumAleatorio(0,3);
    TipoSeleccion=Seleccion[PSele];//tipo de seleccion
}
```

```
/*-------------------------------------------------------------------------
Cada proceso define los parámetros que utilizará para solucionar el problema
-------------------------------------------------------------------------*/
void DefineParametros()
{
    //parametros del algoritmo sobre que porcentaje de la población cruzar o mutar
    int PrCruce[5]={60,70,80,90,100};
    int PrMuta[5]={10,20,30,40,50};
    int TamPobl[10]={100,200,300,400,500,600,700,1000,5000,10000};
    int NGen[10]={100,200,300,400,500,600,1000,5000,10000,20000};
    int NumGen, TamPob;
    int POpera,PPrCr,PPrMu;
    srand((unsigned)time(NULL)+IdProceso);//semilla
    POpera=GeneraNumAleatorio(0,1);
    TamPob=GeneraNumAleatorio(0,9);
    TamPoblacion=TamPobl[TamPob];
    PPrCr=GeneraNumAleatorio(0,4);
    ProbCr=PrCruce[PPrCr];
    if(POpera==1)
    {
        PPrMu=100-PrCruce[PPrCr];
        ProbMuta=PPrMu;
    }
    else
    {
        PPrMu=GeneraNumAleatorio(0,4);
        ProbMuta=PrMuta[PPrMu];
    }
    NumGen=GeneraNumAleatorio(0,9);
    NumGeneraciones=NGen[NumGen];
}


/*-------------------------------------------------------------------------
El maestro almacena las tareas de los alumnos
-------------------------------------------------------------------------*/
void AlmacenaResultados()
{
    float Solucion[NumTuberias];
    float CostoSoluc[NumTuberias];
    int NumAlum,I,J,M,K=0,L=0;
    NumAlum=NumIndivGrupo-1;
    for (I = 0; I < NumAlum; I++){
        for (J = 0; J < NumSolRecibir; J++){
            if(aptitudesTemp[K].CostoTotalConfigRed > 0){
                for (M = 0; M < NumTuberias; M++ )
                {
                    poblacionDescendientes[ContDescendientes].Indice=ContDescendientes;
                    asigna al arreglo Configuracion un diametro disponible a cada posicion
                    poblacionDescendientes[ContDescendientes].ConfigRed[M]=PoblacionTemp[L].ConfigRed[M];
                    se copian los costos para cada diametro
                    poblacionDescendientes[ContDescendientes].CostoConfigRed[M]=PoblacionTemp[L].CostoConfigRed[M];
                }
                se copia el costo total de la nueva configuracion
                aptitudesDescendientes[ContDescendientes].CostoTotalConfigRed=aptitudesTemp[K].CostoTotalConfigRed;
                aptitudesDescendientes[ContDescendientes].Indice=ContDescendientes;
                ContDescendientes++;
            }
            L++;
            K++;
        }
    }
    fclose(fp);
}
```

```
/*--------------------------------------------------------------------------
El alumno envia al maestro los resultados del problema
--------------------------------------------------------------------------*/
void EnviaResultados()
{
    int I=0,J,K=0,L=0,NumElem;
    NumElem=(2*NumTuberias+1)*NumSolEntregar;
    for (I = 0; I < NumSolEntregar; I++){
        L=0;
        for (J = 0; J < NumTuberias; J++){
            Resultado[K]=poblacion[I].ConfigRed[J];
            K++;
        }
        for (J = NumTuberias; J < (2*NumTuberias); J++){
            Resultado[K]=poblacion[I].CostoConfigRed[L];
            L++;
            K++;
        }
        if(J==(2*NumTuberias)){
            Resultado[K]=aptitudes[I].CostoTotalConfigRed;
            K++;
        }
    }
    if(MPI_Send(&Resultado,NumElem, MPI_FLOAT,profe,etiqueta,NuevoCom)== MPI_SUCCESS)
        printf("\n\nNotificacion de envio de resultados enviada correctamente");
    else
        printf("\n\nError de Envio");
}


/*--------------------------------------------------------------------------
El maestro recibe las tareas de los alumnos
--------------------------------------------------------------------------*/
void RecibeResultados()
{
    char nombreArch[20];
    int I,J,K=0,L=0,NumElem,M=0,P=0;
    NumElem=(2*NumTuberias+1)*NumSolRecibir;
    for (K=0; K<NumIndivGrupo; K++) {
        L=M=0;
        if(IdSubProceso != K){
            if(MPI_Recv(&Resultado,NumElem,MPI_FLOAT,K,etiqueta,NuevoCom, &estado) == MPI_SUCCESS){
                for (I = 0; I < NumSolRecibir; I++){
                    for (J = 0; J < NumTuberias; J++){
                        PoblacionTemp[P].ConfigRed[J]=Resultado[L];
                        L++;
                    }
                    for (J = NumTuberias; J < (2*NumTuberias); J++){
                        PoblacionTemp[P].CostoConfigRed[M]=Resultado[L];
                        L++;
                        M++;
                    }
                    if(J==(2*NumTuberias)){
                        aptitudesTemp[P].CostoTotalConfigRed=Resultado[L];
                        L++;
                    }
                    P++;
                }
            }
            else{
                printf("\n\nError de Recepcion en el maestro");
                exit(1);
            }
        }
    }
```

```
/*-------------------------------------------------------------------------
Cuerpo del Algoritmo Evolutivo, aquí se utilizan los operadores genéticos
-------------------------------------------------------------------------*/
void EvaluaResultados()
{
    int i,I,J,L,Z,Indiv=0,Indiv2=0;
    int NumIndividuos=150000,tiempo=0; //es la muestra tomada de la poblacion total
    char nombreArch[20];
    FILE *fp;
    AgendaTE=TiempoEntr;
    sprintf(nombreArch,"ArchResultados%d%s",IdProceso,".txt");
    fp=fopen(nombreArch,"at");
    fprintf(fp,"NumGeneraciones %d \t TamPoblacion %d \t PrCruce %d \t PrMuta %d", NumGeneraciones,TamPoblacion,ProbCr,ProbMuta);
    DefineEstrategiaSolucion();
    fprintf(fp,"\nDEFINE ESTRATEGIA \n");
    fprintf(fp,"Dependientes %d \t Tipo de Cruce1.mono 2multi %d \t TipoSeleccion%d \t \n", POpera,TipoCr,TipoSeleccion);
    ContDescendientes=0;
    if(LeeRDA())
    {
        GeneraPoblacionFactible(NumTuberias,NumIndividuos);
        OrdenaPoblacionFactible(0,(TamPoblacionFactible-1));
        GuardaPoblacionFactibleOrdenada();
        CopiaSubconjunto();
        for(J=0;  J < TamPoblacion; J++)
            CreaDiversidad(J);
        // Número de iteraciones que realizará el algoritmo evolutivo
        // en cada generacion recibe soluciones de los alumnos
        for(I=1;  I <= NumGeneraciones; I++)
        {

            ContDescendientes=0;
            for(J=0;  J < TamPoblacion; J++)
            {
                //Z sirve para decidir mutar o cruzar
                Z=GeneraNumAleatorio(0,100);
                if(Z   <  ProbCr)
                {
                    Indiv=GeneraNumAleatorio(0,(TamPoblacion-1));
                    Indiv2=GeneraNumAleatorio(0,TamPoblacion-1);
                    //cruce monopunto,aleatorio
                    CruzaIndividuos(Indiv,Indiv2);
                }
                Z=GeneraNumAleatorio(0,100);
                if(Z   <  ProbMuta)
                {
                    //Indiv indica cual individuo se mutará
                    Indiv=GeneraNumAleatorio(0,(TamPoblacion-1));
                    MutaIndividuo(Indiv);
                }
            }
            if(AgendaTE == I)
            {
                RecibeNotificacion();
                RecibeResultados();
                AlmacenaResultados();
                AgendaTE=AgendaTE+TiempoEntr;
            }
            //En cada generacion se eligen a los individuos que formarán la nueva población
            OrdenaPobDescendientes(0,(ContDescendientes-1));
            GuardaPobDescOrdenada();
            OrdenaPoblacion(0,(TamPoblacion-1));
            GuardaPoblacionOrdenada();
            RemplazaIndividuos();
            OrdenaPoblacion(0,(TamPoblacion-1));
            GuardaPoblacionOrdenada();
            fprintf(fp,"\n %d \t\t %.2f \t", I, aptitudes[0].CostoTotalConfigRed);
            for (L = 0; L < NumTuberias; L++)
                fprintf(fp, "%.2f-", poblacion[aptitudes[0].Indice].ConfigRed[L]);
        }
        fclose(fp);
    }
```

```
/*-------------------------------------------------------------------------
Cuerpo del Algoritmo Evolutivo, aqui se utilizan los operadores genéticos
-------------------------------------------------------------------------*/
void EvolucionaPoblacion()
{
    int I,J,Z,L,Indiv=0,Indiv2=0,Indiv3=0,Indiv4=0,Ganador1, Ganador2;
    int EnviaResul=1,n=1,NumIndividuos=150000,tiempo=0; //es la muestra tomada de la poblacion total
    FILE *fp,*fpr;
    char nombreArch[20];
    char nombresArch[20];
    AgendaTE=TiempoEntr;
    sprintf(nombreArch,"ArchResultados%d%s",IdProceso,".txt");
    fp=fopen(nombreArch,"at");
    sprintf(nombresArch,"ArchsResultados%d%s",IdProceso,".txt");
    fpr=fopen(nombresArch,"at");
    if(corridas == 0)
    fprintf(fp,"DEFINE PARAMETROS \n");
    fprintf(fp,"NumGeneraciones %d \t TamPoblacion %d \t PrCruce %d \t PrMuta %d", NumGeneraciones,TamPoblacion,ProbCr,ProbMuta);

    ContDescendientes=0;
    DefineEstrategiaSolucion();
    fprintf(fp,"\nDEFINE ESTRATEGIA \n");
    fprintf(fp,"Dependientes %d \t Tipo de Cruce: %d \t TipoSeleccion%d \t", POpera,TipoCr,TipoSeleccion);
    fprintf(fp,"\n");
    if(LeeRDA())
    {
        srand((unsigned)time(NULL)+IdProceso);//semilla
        GeneraPoblacionFactible(NumTuberias,NumIndividuos);
        OrdenaPoblacionFactible(0,(TamPoblacionFactible-1));
        GuardaPoblacionFactibleOrdenada();
        CopiaSubconjunto();
        for(J=0;  J < TamPoblacion; J++)
            CreaDiversidad(J);
        (void)time(&timer1);
        // Número de iteraciones que realizará el algoritmo evolutivo
        for(I=1;  I <= NumGeneraciones; I++)
        {
            ContDescendientes=0;
            for(J=0;  J < TamPoblacion; J++)
            {
                if (Operadores==1)
                {
                    Z=GeneraNumAleatorio(0,100);
                    if(Z   <  ProbCr)
                    {

                        // se hacen diferentes tipos de selección
                        switch(TipoSeleccion)
                        {
                        case 1://selección aleatoria
                            Indiv=GeneraNumAleatorio(0,(TamPoblacion-1));
                            Indiv2=GeneraNumAleatorio(0,TamPoblacion-1);
                            CruzaIndividuos(Indiv,Indiv2);
                        break;

                        case 2://selección torneo
                            Indiv=GeneraNumAleatorio(0,(TamPoblacion-1));
                            Indiv2=GeneraNumAleatorio(0,TamPoblacion-1);
                            Indiv3=GeneraNumAleatorio(0,(TamPoblacion-1));
                            Indiv4=GeneraNumAleatorio(0,TamPoblacion-1);
                            if(aptitudes[Indiv].CostoTotalConfigRed < aptitudes[Indiv2].CostoTotalConfigRed)
                                Ganador1=Indiv;
                            else
                                Ganador1=Indiv2;
                            if(aptitudes[Indiv3].CostoTotalConfigRed < aptitudes[Indiv4].CostoTotalConfigRed)
                                Ganador2=Indiv3;
                            else
                                Ganador2=Indiv4;
                            CruzaIndividuos(Ganador1,Ganador2);

                        break;
                        }
                    }
```

```
            Z=GeneraNumAleatorio(0,100);
            if(Z    <   ProbMuta)
            {
                //Indiv indica cual individuo se mutará
                Indiv=GeneraNumAleatorio(0,(TamPoblacion-1));
                MutaIndividuo(Indiv);
            }
        }
    else
    {
            //Z sirve para decidir mutar o cruzar, de la poblacion total se elige un individuo
            Z=GeneraNumAleatorio(0,100);
            if(Z    <   ProbMuta)
            {
                //Indiv indica cual individuo se mutará
                Indiv=GeneraNumAleatorio(0,(TamPoblacion-1));
                MutaIndividuo(Indiv);
            }
            else
            {
                // se hacen diferentes tipos de selección
                switch(TipoSeleccion)
                {
                case 1://selección aleatoria
                    Indiv=GeneraNumAleatorio(0,(TamPoblacion-1));
                    Indiv2=GeneraNumAleatorio(0,TamPoblacion-1);
                    CruzaIndividuos(Indiv,Indiv2);
                break;

                case 2://selección torneo
                    Indiv=GeneraNumAleatorio(0,(TamPoblacion-1));
                    Indiv2=GeneraNumAleatorio(0,TamPoblacion-1);
                    Indiv3=GeneraNumAleatorio(0,(TamPoblacion-1));
                    Indiv4=GeneraNumAleatorio(0,TamPoblacion-1);
                    if(aptitudes[Indiv].CostoTotalConfigRed < aptitudes[Indiv2].CostoTotalConfigRed)
                        Ganador1=Indiv;
                    else
                        Ganador1=Indiv2;
                    if(aptitudes[Indiv3].CostoTotalConfigRed < aptitudes[Indiv4].CostoTotalConfigRed)
                        Ganador1=Indiv2;
                    if(aptitudes[Indiv3].CostoTotalConfigRed < aptitudes[Indiv4].CostoTotalConfigRed)
                        Ganador2=Indiv3;
                    else
                        Ganador2=Indiv4;
                    CruzaIndividuos(Ganador1,Ganador2);

                break;
                }
            }
        }
    }
    //En cada generacion se eligen a los individuos que formarán la nueva población
OrdenaPobDescendientes(0,(ContDescendientes-1));
GuardaPobDescOrdenada();
OrdenaPoblacion(0,(TamPoblacion-1));
GuardaPoblacionOrdenada();
RemplazaIndividuos();
OrdenaPoblacion(0,(TamPoblacion-1));
GuardaPoblacionOrdenada();
fprintf(fpr,"\n %d \t %.2f", I, aptitudes[0].CostoTotalConfigRed);
fprintf(fp,"\n %d \t\t %.2f \t", I, aptitudes[0].CostoTotalConfigRed);
for (L = 0; L < NumTuberias; L++)
    fprintf(fp, "%.2f-", poblacion[aptitudes[0].Indice].ConfigRed[L]);
for (L = 0; L < NumNodos; L++)
    fprintf(fp, "%.2f-", poblacion[aptitudes[0].Indice].PresionesRed[L]);
//cada vez que deba enviar resultados al maestro
if(AgendaTE == I)
{
    EnviaNotificacion(1);
    EnviaResultados();
    //registra en agenda nueva fecha de entrega de resultados
    AgendaTE=AgendaTE+TiempoEntr;
}
}
```

```
/*-------------------------------------------------------------------------
Obtiene la conectividad de la red
-------------------------------------------------------------------------*/
void ObtieneConectividad()
{
    char *ArchivoEntrada;
    char *ArchivoReporte;
    char *ArchivoSalida;
    int  CodigoError,I,x,y,C=0,D=0;

    ArchivoEntrada="Hanoi.inp";
    ArchivoReporte="Hanoi.txt";
    ArchivoSalida="Hanoi.dat";

    CodigoError=ENopen(ArchivoEntrada, ArchivoReporte,ArchivoSalida);
    if (CodigoError > 0)
    {
        printf("\n NO se pudo abrir archivo\n ");
        ENclose();
    }
    else
    {
        //llamadas a las funciones de epanet
        ENopenH();
        ENgetcount(EN_LINKCOUNT, &NumTuberias);
        ENgetcount(EN_NODECOUNT, &NumNodos);

        for(I=1; I <= NumTuberias; I++)
        {

            ENgetlinknodes(I,&x,&y);    //fuente, destino
            conecta[C].IdTuberia=I;
            conecta[C].origen=x;
            conecta[C].destino=y;
            C ++;
        }
        ENcloseH();
        ENclose();
    }
  //se definen pares (tuberia, conectado) que indica a que nodo está
  //conectada una tuberia
  C=0;
  for(I=1; I <= NumTuberias; I++)
  {
      Conexiones[D].tuberia=conecta[C].IdTuberia;
      Conexiones[D].conectado=conecta[C].origen;
      Conexiones[D+1].tuberia=conecta[C].IdTuberia;
      Conexiones[D+1].conectado=conecta[C].destino;
      D += 2;
      C++;
  }
  //ordena los nodos de la red y muestra que tuberias se conectan a dichos nodos
  OrdenaConexiones(0,D-1);
  GuardaConexionesOrdenadas(D);
}
```

```
/*----------------------------------------------------------------------------
Carga la configuración de la red desde un archivo de entrada
----------------------------------------------------------------------------*/
int LeeRDA(float *Configuracion)
{
    char *ArchivoEntrada;
    char *ArchivoReporte;
    char *ArchivoSalida;
    int  CodigoError;

    ArchivoEntrada="Hanoi.inp";
    ArchivoReporte="Hanoi.txt";
    ArchivoSalida="Hanoi.dat";
    CodigoError=ENopen(ArchivoEntrada, ArchivoReporte,ArchivoSalida);
    if (CodigoError > 0)
    {
        printf("\n NO se pudo abrir archivo\n ");
        ENclose();
        return 0;
    }
    else
    {
        //llamadas a las funciones de epanet
        ENopenH();
        ENgetcount(EN_LINKCOUNT, &NumTuberias);
        ENgetcount(EN_NODECOUNT, &NumNodos);
    }
    return 1;
}

/*----------------------------------------------------------------------------
Modulo de analisis hidraulico.Recibe una configuración a analizar.
Devuelve el valor de NumDebMin que indica el numero de nodos en los cuales
las presiones son menores a las presiones mínimas requeridas.
----------------------------------------------------------------------------*/
int AnalizaRDA(float *Configuracion)
{
    int I,J,tipoNodo,NumDebMin=0,I=0;
    float p,VelocidadTub=0.0;
    long t;
    //asigna a la red una nueva configuración de diámetros de tuberías
    for(I=0;  I < NumTuberias; I++)
        ENsetlinkvalue(I+1, EN_DIAMETER,Configuracion[I]);
    ENinitH(0);
    ENrunH(&t);
    J=0;
    while(J < NumNodos){
        p=0;
        ENgetnodetype(J, &tipoNodo);
        //si es nodo de demanda
        if(tipoNodo == 0){
            //se obtiene presion y se guarda en la variable p
            ENgetnodevalue(J, EN_PRESSURE, &p);
            //Guarda presiones en los nodos de la red que analiza
            if(p < PresionMinReq || p > PresionMaxReq ){
                NumDebMin++;
                J=NumNodos;
            }
            presionesConfig[J]=p;
        }
        J++;
    }
    if(corridas==30){
        ENclose();
        ENcloseH();
    }
    return NumDebMin;
}
```

```
/*-------------------------------------------------------------------------
Evalua una configuración infactible e intenta ajustarla a factible cambiando
tuberias en las que los usuarios no obtienen un servicio adecuado
-------------------------------------------------------------------------*/
int AjustaaFactible(float *Configuracion,float *CostoConfiguracion)
{
    int pos,posN,desicion,i,j,cont=0;
    int PosTubIncidentes[4];
    int tuberiaCambiar;
    int posNodo=0;
    int NumDeb, NumDebNueva;
    float CostosNuevaR[NumTuberias];
    AjustaFact=pos=posN=0;
    NumDeb=NumDebMin;
    if(Ajustes==0)
    for(j=0; j < NumTuberias; j++){
        ConfigNuevaR[j]=Configuracion[j];
        CostosConfigNuevaR[j]=CostoConfiguracion[j];
    }
    CostoNuevaRed=CostoConfigR;
    for(j=0; j < NumTuberias; j++){
        ConfigTmp[j]=Configuracion[j];
        CostosConfigTmp[j]=CostoConfiguracion[j];
    }
    for(i=0; i < NumNodos; i++){
    //se revisa cual nodo no alcanza presion mínima
        if(presionesConfig[i] < PresionMinReq)
        {
            //se debe saber que tuberias inciden en el nodo cuya presión esta debajo de la mínima requerida
            posNodo=BuscaConexionesNodo(SumaGrados,i);//realmente debe ir el numero de conexiones
            //rango de tuberias conectadas a un nodo
            //suponiendo que un nodo tiene 4 aristas incidentes (maximo grado)
            if(posNodo>=2 && posNodo<=SumaGrados-2)
                for(j=posNodo-2; j<posNodo+2; j++){
                    if(Conexiones[j].conectado == i){
                        PosTubIncidentes[cont]=j;
                        cont++;
                    }
                }
        //se genera num aleatorio para decidir si se cambia el gen de forma deterministica o aleatoria
        desicion=GeneraNumAleatorio(0, 100);
        tuberiaCambiar=Conexiones[posNodo].tuberia;
        if(desicion < 50){
            //Identifica la posición del diametro actual dentro del vector tuberias y se cambia el diámetro por otro más grande
            //dice en que posición de diametros comerciales esta la tuberia a cambiar.
            pos= BuscaTuberia(Configuracion,tuberiaCambiar);
            posN=GeneraNumAleatorio((pos),(NumDiametros-1));
            ConfigTmp[tuberiaCambiar]=DiametrosComerciales[posN];
            CostosConfigTmp[tuberiaCambiar]=CostoDiametros[posN];
        }
        else{
            // Se cambia el diametro que no cumple por otro de forma aleatoria
            posN=GeneraNumAleatorio(0, (NumDiametros-1));
            ConfigTmp[tuberiaCambiar]=DiametrosComerciales[posN];
            CostosConfigTmp[tuberiaCambiar]=CostoDiametros[posN];
        }
    }
}
Ajustes++;
NumDebNueva=AnalizaRDA(ConfigTmp);
if(NumDebNueva==0)   {
    for(j=0; j < NumTuberias; j++){
        ConfigNuevaR[j]=ConfigTmp[j];
        CostosConfigNuevaR[j]=CostosConfigTmp[j];
        CostoNuevaRed = CostoNuevaRed + (CostosConfigTmp[j]*LongitudTuberia[j]);
    }
}
else {
    if(Ajustes  < 20 && AjustaFact!=1){
        if(NumDebNueva < NumDeb){
            for(j=0; j < NumTuberias; j++){
                ConfigNuevaR[j]=ConfigTmp[j];
                CostosConfigNuevaR[j]=CostosConfigTmp[j];
                CostoNuevaRed = CostoNuevaRed + (CostosConfigTmp[j]*LongitudTuberia[j]);
            }
            AjustaaFactible(ConfigNuevaR,CostosConfigNuevaR);
        }

        Ajustes++;
    }
}
return AjustaFact;
}
```

```
/*-------------------------------------------------------------------------
Obtiene el costo de una tuberia de acuerdo a su longitud y a su diametro.
-------------------------------------------------------------------------*/
float ObtieneCosto(int i)
{
    float CostoTub;
    CostoTub=CostoDiametros[i]*LongitudTuberia[i];
    return CostoTub;
}

/*-------------------------------------------------------------------------
Calcula el costo de una configuracion de acuerdo a su longitud y a su diametro.
-------------------------------------------------------------------------*/
float CalculaCosto(int Indv)
{
    int pos,i;
    float CostoConfiguracion=0;
    for (i = 0; i < NumTuberias; i++ )
    {
        pos = BuscaElemento(Indv,i);
        poblacion[Indv].CostoConfigRed[i]=CostoDiametros[pos];
        CostoConfiguracion += ((CostoDiametros[pos])*LongitudTuberia[i]);
    }
    aptitudes[Indv].CostoTotalConfigRed=CostoConfiguracion;
    return CostoConfiguracion;
}

/*-------------------------------------------------------------------------
Calcula el costo de una configuracion de acuerdo a su longitud y a su diametro.
----- -------------------------------------------------------------------------*/
float CalculaCostoConfig(float *Config)
{
    float CostoConfiguracion=0;
    int pos=0,i,bajo=0,medio;
    int alto=NumDiametros-1;
    int encontrado=0;
    for (i = 0; i < NumTuberias; i++ )
    {
        encontrado=0;
        while(bajo <=alto && !encontrado)
        {
            medio=(bajo+alto)/2;
            if(Config[i] == DiametrosComerciales[medio])
            {
                pos= medio;
                encontrado=1;
            }
            else if(Config[i] < DiametrosComerciales[medio])
                alto=medio-1;
            else
                bajo=medio+1;
        }
        CostoConfiguracion += ((CostoDiametros[pos])*LongitudTuberia[i]);
    }
return CostoConfiguracion;
}
```

```
/*-------------------------------------------------------------------------
Ordena a los individuos en forma ascendente usando el método quicksort cuya
complejidad computacional es O(nlogn).
Coloca primero a los individuos de menor costo (más aptos) y conforme avanza
el indice coloca a los individuos de mayor costo(menos aptos).
-------------------------------------------------------------------------*/
void OrdenaPoblacion(int primero, int ultimo)
{
    int posicionActual;
    if(primero >= ultimo)
        return;
    posicionActual=ParticionaArreglo(primero, ultimo);
    OrdenaPoblacion(primero, posicionActual -1);
    OrdenaPoblacion(posicionActual+1, ultimo);

}

int ParticionaArreglo(int izquierda, int derecha)
{
    int posicion, aux,aux2;
    aux=0;
    aux2=0;
    posicion=izquierda;
    while(True)
    {
        while (aptitudes[posicion].CostoTotalConfigRed <= aptitudes[derecha].CostoTotalConfigRed && posicion != derecha)
            derecha--;
            if(posicion == derecha)
                return posicion;
            if (aptitudes[posicion].CostoTotalConfigRed > aptitudes[derecha].CostoTotalConfigRed)
            {
                aux=aptitudes[posicion].CostoTotalConfigRed;
                aux2=aptitudes[posicion].Indice;
                aptitudes[posicion].CostoTotalConfigRed=aptitudes[derecha].CostoTotalConfigRed;
                aptitudes[posicion].Indice=aptitudes[derecha].Indice;
                aptitudes[derecha].CostoTotalConfigRed=aux;
                aptitudes[derecha].Indice=aux2;
                posicion=derecha;
            }
            while(aptitudes[izquierda].CostoTotalConfigRed <= aptitudes[posicion].CostoTotalConfigRed && izquierda != posicion)
                izquierda ++;
            if(posicion == izquierda)
                return posicion;
            if(aptitudes[izquierda].CostoTotalConfigRed > aptitudes[posicion].CostoTotalConfigRed)
            {
                aux=aptitudes[posicion].CostoTotalConfigRed;
                aux2=aptitudes[posicion].Indice;
                aptitudes[posicion].CostoTotalConfigRed=aptitudes[izquierda].CostoTotalConfigRed;
                aptitudes[posicion].Indice=aptitudes[izquierda].Indice;
                aptitudes[izquierda].CostoTotalConfigRed=aux;
                aptitudes[izquierda].Indice=aux2;
                posicion=izquierda;
            }
    }

}
```

```
/*-------------------------------------------------------------------------
Guarda la nueva población ordenada en un la estructura población
-------------------------------------------------------------------------*/
void GuardaPoblacionOrdenada()
{
    int I,J;
    for (I=0; I < TamPoblacion; I++)
    {
        for (J = 0; J < NumTuberias; J++)
        {
            poblacionAuxiliar[I].ConfigRed[J]=poblacion[aptitudes[I].Indice].ConfigRed[J];
            poblacionAuxiliar[I].CostoConfigRed[J]=poblacion[aptitudes[I].Indice].CostoConfigRed[J];
            poblacionAuxiliar[I].PresionesRed[J]=poblacion[aptitudes[I].Indice].PresionesRed[J];
            poblacionAuxiliar[I].RestriccionesHid=poblacion[aptitudes[I].Indice].RestriccionesHid;
            poblacionAuxiliar[I].AjustaPresion=poblacion[aptitudes[I].Indice].AjustaPresion;
            poblacionAuxiliar[I].NumDebMin=poblacion[aptitudes[I].Indice].NumDebMin;
        }
        printf("\n");
    }
    for (I = 0 ; I < TamPoblacion; I++)
    {
        for (J = 0; J < NumTuberias; J++)
        {
            // se actualizan los indices de la estructura aptitudes
            aptitudes[I].Indice=I;
            poblacion[I].ConfigRed[J]=poblacionAuxiliar[I].ConfigRed[J];
            //Se regresan los datos de la estructura auxiliar a la estructura poblacion
            poblacion[I].CostoConfigRed[J]=poblacionAuxiliar[I].CostoConfigRed[J];
            poblacion[I].PresionesRed[J]=poblacionAuxiliar[I].PresionesRed[J];
            poblacion[I].RestriccionesHid=poblacionAuxiliar[I].RestriccionesHid;
            poblacion[I].AjustaPresion=poblacionAuxiliar[I].AjustaPresion;
            poblacion[I].NumDebMin=poblacionAuxiliar[I].NumDebMin;
        }
    }

/*-------------------------------------------------------------------------
Búsqueda Binaria. Dado un elemento, lo busca en un vector y devuelve la
posición en la que se encuentra.
-------------------------------------------------------------------------*/
int BuscaTuberia(float *Config,int tuberia)
{
    int bajo=0;
    int alto=NumDiametros-1;//es el arreglo de numero de diametros comerciales
    int medio;
    int i;
    int encontrado=0;

    for (i = 0; i < NumTuberias; i++ )
    {
        encontrado=0;
        while(bajo <=alto && !encontrado)
        {
            medio=(bajo+alto)/2;
            if(Config[i] == DiametrosComerciales[medio])
            {
                encontrado=1;
                return medio;
            }
            else if(Config[i] < DiametrosComerciales[medio])
                alto=medio-1;
            else
                bajo=medio+1;
        }
    }
    return -1;
}
```

```
/*-------------------------------------------------------------------------
Cruzamiento.Consiste en combinar dos individuos para obtener descendientes.
Se elige un punto de cruce, para los 2 individuos,  de forma aleatoria.
Al dividir cada individuo en dos partes, se tienen cuatro partes: A, B, C y D
Se combina A y D y generan el primer descendiente.
Se combina C y B y se tiene el segundo descendiente.
-------------------------------------------------------------------------*/
void CruzaIndividuos(int indiv1, int indiv2)
{
    int i,j;
    float c1,c2;
    float Individuo1[NumTuberias];
    float Individuo2[NumTuberias];
    float CostoI1a, CostoI1b, CostoI1c,CostoI1d;
    float CostoI2a, CostoI2b, CostoI2c, CostoI2d;
    int puntoCruce2, puntoCruce;

    CostoI1a=CostoI1b=CostoI1c=CostoI1d=0;
    CostoI2a=CostoI2b=CostoI2c=CostoI2d=0;
    for(i=0; i < NumTuberias ; i++){
        CostoDiamI1[i]=0;
        CostoDiamI2[i]=0;
    }
    // tipo de cruce: monopunto
    if (TipoCr==1){
        //punto de cruce aleatorio
        if (PuntoCr==1)
            puntoCruce=GeneraNumAleatorio(0, NumTuberias-1);
        else
        //punto de cruce a la mitad del cromosoma
            puntoCruce=NumTuberias/2;
        for(i=0; i < puntoCruce; i++){
            // Se guardan los descendientes
            Individuo1[i]=poblacion[indiv1].ConfigRed[i];
            Individuo2[i]=poblacion[indiv2].ConfigRed[i];
            // Se guarda el costo correspondiente a cada tuberia
            CostoDiamI1[i]=poblacion[indiv1].CostoConfigRed[i];
            CostoDiamI2[i]=poblacion[indiv2].CostoConfigRed[i];
            // Se guarda el costo total del individuo
           CostoI1a += poblacion[indiv1].CostoConfigRed[i]*LongitudTuberia[i];
           CostoI2a += poblacion[indiv2].CostoConfigRed[i]*LongitudTuberia[i];
       }
       for(j=puntoCruce; j < NumTuberias; j++){
           //Se guardan los descendientes
           Individuo1[j]=poblacion[indiv2].ConfigRed[j];
           Individuo2[j]=poblacion[indiv1].ConfigRed[j];
           //Se guarda el costo correspondiente a cada tuberia
           CostoDiamI1[j]= poblacion[indiv2].CostoConfigRed[j];
           CostoDiamI2[j]=poblacion[indiv1].CostoConfigRed[j];
           //Se guarda el costo total de cada individuo
           CostoI1b += poblacion[indiv2].CostoConfigRed[j]*LongitudTuberia[j];
           CostoI2b += poblacion[indiv1].CostoConfigRed[j]*LongitudTuberia[j];
       }

   }
```

```
//tipo de cruce:multipunto
else{
//punto de cruce aleatorio
    if (PuntoCr==1){
        puntoCruce=GeneraNumAleatorio(0, NumTuberias-1);
        puntoCruce2=GeneraNumAleatorio(0, puntoCruce-1);

    }
    else{
    //punto de cruce a la mitad del cromosoma
        puntoCruce=NumTuberias/2;
        puntoCruce2=puntoCruce/2;
    }
    for(i=0; i < puntoCruce2; i++)
    {
        // Se guardan los descendientes
        Individuo1[i]=poblacion[indiv1].ConfigRed[i];
        Individuo2[i]=poblacion[indiv2].ConfigRed[i];
        // Se guarda el costo correspondiente a cada tuberia
        CostoDiamI1[i]=poblacion[indiv1].CostoConfigRed[i];
        CostoDiamI2[i]=poblacion[indiv2].CostoConfigRed[i];
        // Se guarda el costo total del individuo
        CostoI1a += poblacion[indiv1].CostoConfigRed[i]*LongitudTuberia[i];
        CostoI2a += poblacion[indiv2].CostoConfigRed[i]*LongitudTuberia[i];
    }
    for(i=puntoCruce2; i < puntoCruce; i++){
        // Se guardan los descendientes
        Individuo1[i]=poblacion[indiv1].ConfigRed[i];
        Individuo2[i]=poblacion[indiv2].ConfigRed[i];
        // Se guarda el costo correspondiente a cada tuberia
        CostoDiamI1[i]=poblacion[indiv1].CostoConfigRed[i];
        CostoDiamI2[i]=poblacion[indiv2].CostoConfigRed[i];
        // Se guarda el costo total del individuo
        CostoI1b += poblacion[indiv1].CostoConfigRed[i]*LongitudTuberia[i];
        CostoI2b += poblacion[indiv2].CostoConfigRed[i]*LongitudTuberia[i];
    }
    for(j=puntoCruce; j < puntoCruce + puntoCruce2; j++){
        //Se guardan los descendientes
        Individuo1[j]=poblacion[indiv2].ConfigRed[j];
        Individuo2[j]=poblacion[indiv1].ConfigRed[j];
        //Se guarda el costo correspondiente a cada tuberia
        CostoDiamI1[j]= poblacion[indiv2].CostoConfigRed[j];
        CostoDiamI2[j]=poblacion[indiv1].CostoConfigRed[j];
        //Se guarda el costo total de cada individuo
        CostoI1c += poblacion[indiv2].CostoConfigRed[j]*LongitudTuberia[j];
        CostoI2c += poblacion[indiv1].CostoConfigRed[j]*LongitudTuberia[j];
    }
    for(j=puntoCruce; j < NumTuberias; j++) {
        //Se guardan los descendientes
        Individuo1[j]=poblacion[indiv2].ConfigRed[j];
        Individuo2[j]=poblacion[indiv1].ConfigRed[j];
        //Se guarda el costo correspondiente a cada tuberia
        CostoDiamI1[j]= poblacion[indiv2].CostoConfigRed[j];
        CostoDiamI2[j]=poblacion[indiv1].CostoConfigRed[j];
        //Se guarda el costo total de cada individuo
        CostoI1d += poblacion[indiv2].CostoConfigRed[j]*LongitudTuberia[j];
        CostoI2d += poblacion[indiv1].CostoConfigRed[j]*LongitudTuberia[j];
```

```
            CostoI2d += poblacion[indiv1].CostoConfigRed[j]*LongitudTuberia[j];
        }
    }
    c1=CostoI1a+CostoI1b+CostoI1c+CostoI1d;
    c2=CostoI2a+CostoI2b+CostoI2c+CostoI2d;
    //Se insertan los descendientes en la nueva poblacion
    if (AnalizaRDA(Individuo1)==0){
        GuardaDescendiente(Individuo1,CostoDiamI1,c1);
    }
    else{
        if(poblacion[indiv1].NumDebMin < NumTuberias/2)
            if (AjustaaFactible(Individuo1,CostoDiamI1)==1){
                GuardaDescendiente(ConfigNuevaR,CostosConfigNuevaR,CostoNuevaRed);
            }
    }
    if (AnalizaRDA(Individuo2)==0){
        GuardaDescendiente(Individuo2,CostoDiamI2,c2);
    }
    else{
        if(poblacion[indiv2].NumDebMin < NumTuberias/5)
        if (AjustaaFactible(Individuo2,CostoDiamI2)==1){
            GuardaDescendiente(ConfigNuevaR,CostosConfigNuevaR,CostoNuevaRed);
        }
    }
}
```

```
/*-------------------------------------------------------------------------
Mutación(es un pequeño cambio): el cambio puede ser aumentar un gen,
disminuir un gen o bien cambiarlo de forma aleatoria por alguno de los
diametros de tuberias disponibles.
Recibe como parámetro el numero de individuo que se mutará y devuelve un nuevo
individuo mutado.
-------------------------------------------------------------------------*/
void MutaIndividuo(int Indiv)
{
    int Tipo,Tuberia;
    int pos,cont,i,I,Z=0,muta;
    float auxiliar[NumTuberias];
    float CostoIndiv[NumTuberias];
    float costo;
    cont=0;
    costo=0;

    for(i=0; i < NumTuberias; i++)
    {
        //Se copia el individuo a mutar en un arreglo auxiliar
        auxiliar[i]=poblacion[Indiv].ConfigRed[i];
        // Se guarda el costo correspondiente a cada tuberia
        CostoIndiv[i]=poblacion[Indiv].CostoConfigRed[i];
    }
    for(I=0; I < NumTuberias; I++)
    {
        Z=GeneraNumAleatorio(0,10);
        if(Z <= ProbMut)
        {
            Tuberia=GeneraNumAleatorio(0,NumTuberias-1);    //tuberia del individuo a mutar
            //obtiene posicion, en el arreglo diametros comerciales, de la tuberia a cambiar
            pos=BuscaElemento(Indiv,Tuberia);
            Tipo=GeneraNumAleatorio(1,3);
            switch(Tipo)
            {
                //se modifica una tuberia. se genera un numero aleatorio para ver si se aumenta o disminuye un diámetro
                case 1://aumenta el gen
                    if(pos == NumDiametros-1) //comprueba que no salga del rango de posiciones de diametros de tuberias
                        pos=pos-1;
                    auxiliar[Tuberia]=DiametrosComerciales[pos+1];
                    CostoIndiv[Tuberia]=CostoDiametros[pos+1];
                break;
                case 2://disminuye el gen
                    if(pos  == 0) //comprueba que no salga del rango de posiciones de diametros de tuberias
                        pos=pos+1;
                    auxiliar[Tuberia]=DiametrosComerciales[pos-1];
                    CostoIndiv[Tuberia]=CostoDiametros[pos-1];
                break;
                case 3://se cambia el diametro de forma aleatoria
                    muta=GeneraNumAleatorio(0,NumDiametros-1);
                    auxiliar[Tuberia]=DiametrosComerciales[muta];
                    CostoIndiv[Tuberia]=CostoDiametros[muta];
                break;
            }
        }
    }
    //Calcula el costo de la configuracion mutada
    for(i=0; i < NumTuberias; i++)
    {
        costo += ((CostoIndiv[i])*LongitudTuberia[i]);
    }
    //Se inserta el descendiente en la nueva poblacion
    if (AnalizaRDA(auxiliar)==0)
    {
        GuardaDescendiente(auxiliar,CostoIndiv,costo);
    }
    else
    {
      if (AjustaaFactible(auxiliar,CostoIndiv)==0)
          GuardaDescendiente(auxiliar,CostoIndiv,costo);
    }
}
```

```
/*-------------------------------------------------------------------------
 Algoritmo Evolutivo: Remplaza la poblacion anterior por una nueva población
 -------------------------------------------------------------------------*/
void RemplazaIndividuos()
{
    int I,J,S,j=0,M,N,R;
    //se remplaza la poblacion completa
    if (TipoRemplazo ==1){
        switch(Tipo){
            case 1://Remplazo Elitista
            //se eliminan a todos los padres y los MEJORES descendientes constituyen una nueva poblacion
            for (I=0; I < ContDescendientes; I++){
                for (J = 0; J < NumTuberias; J++){
                    poblacionAuxiliar[j].Indice=I;
                    poblacionAuxiliar[j].ConfigRed[J]=poblacionDescendientes[I].ConfigRed[J];
                    poblacionAuxiliar[j].CostoConfigRed[J]=poblacionDescendientes[I].CostoConfigRed[J];
                    poblacionAuxiliar[j].PresionesRed[J]=poblacionDescendientes[I].PresionesRed[J];
                }
                poblacionAuxiliar[j].RestriccionesHid=poblacionDescendientes[I].RestriccionesHid;
                poblacionAuxiliar[j].AjustaPresion=poblacionDescendientes[I].AjustaPresion;
                poblacionAuxiliar[j].NumDebMin=poblacionDescendientes[I].NumDebMin;
                aptitudesAuxiliar[j].CostoTotalConfigRed=aptitudesDescendientes[I].CostoTotalConfigRed;
                j++;
            }
            GuardaIndividuosSeleccionados();
            break;
           //Remplazo Aleatorio sin ordenar descendientes
            case 2:
            for (I=0; I < TamPoblacion; I++){
                S=GeneraNumAleatorio(0, ContDescendientes);
                for (J = 0; J < NumTuberias; J++){
                    poblacion[I].Indice=I;
                    poblacion[I].ConfigRed[J]=poblacionDescendientes[S].ConfigRed[J];
                    poblacion[I].CostoConfigRed[J]=poblacionDescendientes[S].CostoConfigRed[J];
                    poblacion[I].PresionesRed[J]=poblacionDescendientes[S].PresionesRed[J];
                }
                poblacion[I].RestriccionesHid=poblacionDescendientes[S].RestriccionesHid;
                poblacion[I].AjustaPresion=poblacionDescendientes[S].AjustaPresion;
                poblacion[I].NumDebMin=poblacionDescendientes[S].NumDebMin;
                aptitudes[poblacion[I].Indice].CostoTotalConfigRed=aptitudesDescendientes[S].CostoTotalConfigRed;
            }
            break;
        }//switch
}//if
//se remplaza parte de la poblacion
else{
    //En caso de no saber que porcentaje de individuos fallecen
    if (TasaMortalidad == 1){
        //Define que porcentaje remplazar
        R=GeneraNumAleatorio(0, TamPoblacion);
        for (I=0; I < R; I++){
            M=GeneraNumAleatorio(0, TamPoblacion);
            N=GeneraNumAleatorio(0, ContDescendientes);
            for (J = 0; J < NumTuberias; J++){
                poblacion[M].Indice=I;
                poblacion[M].ConfigRed[J]=poblacionDescendientes[N].ConfigRed[J];
                poblacion[M].CostoConfigRed[J]=poblacionDescendientes[N].CostoConfigRed[J];
                poblacion[M].PresionesRed[J]=poblacionDescendientes[N].PresionesRed[J];
            }
            poblacion[M].RestriccionesHid=poblacionDescendientes[N].RestriccionesHid;
            poblacion[M].AjustaPresion=poblacionDescendientes[N].AjustaPresion;
            poblacion[M].NumDebMin=poblacionDescendientes[N].NumDebMin;
            aptitudes[poblacion[M].Indice].CostoTotalConfigRed=aptitudesDescendientes[N].CostoTotalConfigRed;
        }
    }
    else{
        //la mitad de los individuos son remplazados
        for (I=0; I < TamPoblacion/2; I++){
            M=GeneraNumAleatorio(0, TamPoblacion);
            N=GeneraNumAleatorio(0, ContDescendientes);
            for (J = 0; J < NumTuberias; J++){
                poblacion[M].Indice=I;
                poblacion[M].ConfigRed[J]=poblacionDescendientes[N].ConfigRed[J];
                poblacion[M].CostoConfigRed[J]=poblacionDescendientes[N].CostoConfigRed[J];
                poblacion[M].PresionesRed[J]=poblacionDescendientes[N].PresionesRed[J];
            }
            poblacion[M].RestriccionesHid=poblacionDescendientes[N].RestriccionesHid;
            poblacion[M].AjustaPresion=poblacionDescendientes[N].AjustaPresion;
            poblacion[M].NumDebMin=poblacionDescendientes[N].NumDebMin;
            aptitudes[poblacion[M].Indice].CostoTotalConfigRed=aptitudesDescendientes[N].CostoTotalConfigRed;
        }
    }
}
}
```

```
/*-------------------------------------------------------------------------
Realiza cambios genéticos en un individuo
-------------------------------------------------------------------------*/
void CreaDiversidad(int I)
{
    int i,Tipo,Tuberia;
    int pos,cont,j,muta;
    float costo;
    cont=0;
    costo=0;
    float auxiliar[NumTuberias];
    float CostoIndiv[NumTuberias];

    for(i=0; i < NumTuberias; i++)
    {
        //Se copia el individuo a mutar en un arreglo auxiliar
        auxiliar[i]=poblacionFactible[I].ConfigRed[i];
        // Se guarda el costo correspondiente a cada tuberia
        CostoIndiv[i]=poblacionFactible[I].CostoConfigRed[i];
    }

    for(j=0; j < (NumTuberias*.8); j++)
    {

        Tuberia=GeneraNumAleatorio(0,NumTuberias-1);     //tuberia del individuo a mutar
        //obtiene posicion, en el arreglo diametros comerciales, de la tuberia a cambiar
        pos=BuscaTuberia(auxiliar,Tuberia);
        Tipo=GeneraNumAleatorio(1,3);
        switch(Tipo)
        {
            case 1://disminuye el gen
                if(pos  == 0) //comprueba que no salga del rango de posiciones de diametros de tuberias
                    pos=pos-1;
                auxiliar[Tuberia]=DiametrosComerciales[pos-1];
                CostoIndiv[Tuberia]=CostoDiametros[pos-1];
            break;

             case 2://se cambia la tuberia por un diametro tomado de la primera mitad del vector Diametros comerciales
                muta=GeneraNumAleatorio(0,(NumDiametros-1)/2);
                auxiliar[Tuberia]=DiametrosComerciales[muta];
                CostoIndiv[Tuberia]=CostoDiametros[muta];
             break;

             case 3://se cambia la tuberia por un diametro tomado de la segunda mitad del vector Diametros comerciales
                muta=GeneraNumAleatorio(NumDiametros/2,NumDiametros-1);
                auxiliar[Tuberia]=DiametrosComerciales[muta];
                CostoIndiv[Tuberia]=CostoDiametros[muta];
             break;
         }
     }
     //Calcula el costo de la configuracion mutada
     for(i=0; i < NumTuberias; i++)
     {
         costo += ((CostoIndiv[i])*LongitudTuberia[i]);
     }
     //Se inserta el descendiente en la nueva poblacion
     if (AnalizaRDA(auxiliar)==0)
     {
         GuardaDescendiente(auxiliar,CostoIndiv,costo);
     }
 }
```

Código del Analizador SQL-EAWDND

```
--sp_configure 'show advanced options', 1
-- reconfigure
--sp_configure 'Ad Hoc Distributed Queries', 1
--reconfigure
-- C:\Excel\EXCEL\ArchsResultados0.xls

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER off
GO

Declare @NumArchivo     Int
Declare @NomArchivo     Varchar(150)
Declare @RutaArchivo    Varchar(250)
Declare @StringSQL      Varchar(MAX)

--// Tabla de Informacion
Create Table #tInformacionArchivo
(
    IdElemento      Int Identity,
    IdGeneracion    Int,
    Valor           Decimal(18,3),
    Bloque          Varchar(50),
    NomArchivo      Varchar(200)
)

Create Table #tInformacionArchivoMax
(
    NomArchivo      Varchar(200),
    Bloque          Varchar(50),
    IdGeneracion    Int,
    Mayor           Decimal(18,3)
)

Create Table #tInformacionArchivoMin
(
    NomArchivo      Varchar(200),
    Bloque          Varchar(50),
    IdGeneracion    Int,
    Menor           Decimal(18,3)
)

--// Copiamos los archivos a BD
-- Drop Table #tInformacionArchivo,#tInformacionArchivoMax, #tInformacionArchivoMin

Set @NumArchivo = 0
While @NumArchivo < 6
Begin

    Set @NomArchivo     = 'ArchsResultados' + Convert(Varchar(5),@NumArchivo) + '.xls'

    Set @StringSQL = "Insert    Into #tInformacionArchivo (IdGeneracion, Valor)"
    Set @StringSQL = @StringSQL + " Select  * "
    Set @StringSQL = @StringSQL + " From    OpenRowSet( 'Microsoft.Jet.OLEDB.4.0', "
    Set @StringSQL = @StringSQL + "         'Excel 8.0;Database=D:\Archivos\AlperovitsMayo\30mayo\" + @NomArchivo + "'," 
    Set @StringSQL = @StringSQL + "                 'SELECT * FROM [Hoja1$] Where Valor Is Not Null'    ) "

    Execute(@StringSQL)

    Update  #tInformacionArchivo
    Set     NomArchivo  =   @NomArchivo
    Where   NomArchivo Is Null

    Set @NumArchivo = @NumArchivo + 1
End

Declare @Elemento   Int
Declare @TamBloques Int
Declare @Bloque     Int

Set @Elemento   = 1
Set @Bloque     = 1
Set @TamBloques = 100
```

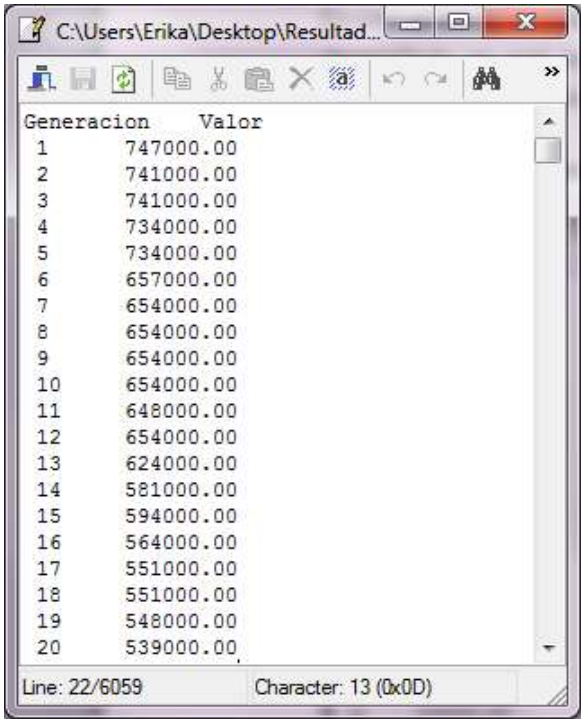
```
-- Update   #tInformacionArchivo Set Bloque = Null

--// Marcamos los datos para poder ordenar
While   Exists(Select * From #tInformacionArchivo Where Bloque Is Null)
Begin
    Update  #tInformacionArchivo
    Set     Bloque      = Convert(Varchar(5), @Bloque)
    Where   IdElemento  = @Elemento

    If  (@Elemento%@TamBloques) = 0
        Set @Bloque = @Bloque + 1
    Set @Elemento   = @Elemento + 1
End

--select * from #tInformacionArchivo

-- Ordenamos los archivos x Bloque
Select  NomArchivo,
        Bloque      = 'BLOQUE ' + Bloque,
        Generacion  = IdGeneracion,
        Orden       = Valor
From    #tInformacionArchivo
Order by NomArchivo, Convert(Int,Bloque) Asc, Valor Asc

-- Obtenemos el menor de cada Bloque
Select  T.NomArchivo,
        Bloque      = 'BLOQUE ' + T.Bloque,
        Generacion  = T.IdGeneracion,
        Minimo      = T.Valor
From    (   Select  B.NomArchivo,
                    Bloque          = B.Bloque,
                    IdGeneracion    = Min(B.IdGeneracion)
            From    (
                        Select  NomArchivo,
                                Bloque      = Bloque,
                                Valor       = Min(Valor)
                        From    #tInformacionArchivo
                        Group by NomArchivo, Bloque
                    ) As A
                    Inner Join #tInformacionArchivo as B
                    On  B.Bloque            = A.Bloque
                        And B.Valor         = A.Valor
                        And B.NomArchivo    = A.NomArchivo
            Group   By B.NomArchivo, B.Bloque
        )   As O
        Inner Join  #tInformacionArchivo as T
        On T.Bloque             = O.Bloque
            And T.IdGeneracion  = O.IdGeneracion
            And T.NomArchivo    = O.NomArchivo
  Order by T.NomArchivo, Convert(Int,T.Bloque) Asc

  -- Obtenemos el dato Menor y mayor por archivo
  Set @NumArchivo = 0
  While @NumArchivo < 6
  Begin

      Set @NomArchivo     = 'ArchsResultados' + Convert(Varchar(5),@NumArchivo) + '.xls'

      -- Obtenemos el Bloque Menor
      Insert  Into #tInformacionArchivoMin
      Select  TOP 1 @NomArchivo,
              Bloque  = 'BLOQUE ' + T.Bloque,
              Generacion      = T.IdGeneracion,
              T.Valor
      From    (   Select  Bloque          = B.Bloque,
                          IdGeneracion    = Min(B.IdGeneracion)
                  From    (
                              Select  Bloque      = Bloque,
                                      Valor       = Min(Valor)
                              From    #tInformacionArchivo
                              Where   NomArchivo  = @NomArchivo
                              Group by Bloque
                          ) As A
                          Inner Join #tInformacionArchivo as B
                          On  B.Bloque        = A.Bloque
                              And B.Valor     = A.Valor
                              And B.NomArchivo= @NomArchivo
```

```
                        On  B.Bloque        = A.Bloque
                            And B.Valor     = A.Valor
                            And B.NomArchivo= @NomArchivo
                Group   By B.Bloque
            )   As O
            Inner Join  #tInformacionArchivo as T
            On T.Bloque             = O.Bloque
                And T.IdGeneracion  = O.IdGeneracion
                And T.NomArchivo    = @NomArchivo
    Order by T.Valor Asc, Convert(Int,T.Bloque) Asc

    -- Obtenemos el Bloque Mayor
    Insert  Into #tInformacionArchivoMax
    Select  TOP 1 @NomArchivo,
            Bloque      = 'BLOQUE ' + T.Bloque,
            Generacion  = T.IdGeneracion,
            Maximo      = T.Valor
    From    (   Select  Bloque          = B.Bloque,
                        IdGeneracion    = Max(B.IdGeneracion)
                From    (
                            Select  Bloque      = Bloque,
                                    Valor       = Max(Valor)
                            From    #tInformacionArchivo
                            Where   NomArchivo  = @NomArchivo
                            Group by Bloque
                        ) As A
                        Inner Join #tInformacionArchivo as B
                        On  B.Bloque        = A.Bloque
                            And B.Valor     = A.Valor
                            And B.NomArchivo= @NomArchivo
                Group   By B.Bloque
            )   As O
            Inner Join  #tInformacionArchivo as T
            On T.Bloque             = O.Bloque
                And T.IdGeneracion  = O.IdGeneracion
                And T.NomArchivo    = @NomArchivo
     Order by T.Valor Asc, Convert(Int,T.Bloque) Asc

     -- Obtenemos el Bloque Mayor
     Insert  Into #tInformacionArchivoMax
     Select  TOP 1 @NomArchivo,
             Bloque      = 'BLOQUE ' + T.Bloque,
             Generacion  = T.IdGeneracion,
             Maximo      = T.Valor
     From    (   Select  Bloque          = B.Bloque,
                         IdGeneracion    = Max(B.IdGeneracion)
                 From    (
                             Select  Bloque      = Bloque,
                                     Valor       = Max(Valor)
                             From    #tInformacionArchivo
                             Where   NomArchivo  = @NomArchivo
                             Group by Bloque
                         ) As A
                         Inner Join #tInformacionArchivo as B
                         On  B.Bloque        = A.Bloque
                             And B.Valor     = A.Valor
                             And B.NomArchivo= @NomArchivo
                 Group   By B.Bloque
             )   As O
             Inner Join  #tInformacionArchivo as T
             On T.Bloque             = O.Bloque
                 And T.IdGeneracion  = O.IdGeneracion
                 And T.NomArchivo    = @NomArchivo
     Order by T.Valor Desc, Convert(Int,T.Bloque) Asc

     Set @NumArchivo = @NumArchivo + 1
End

-- Bloques menores
Select  * From #tInformacionArchivoMin
```

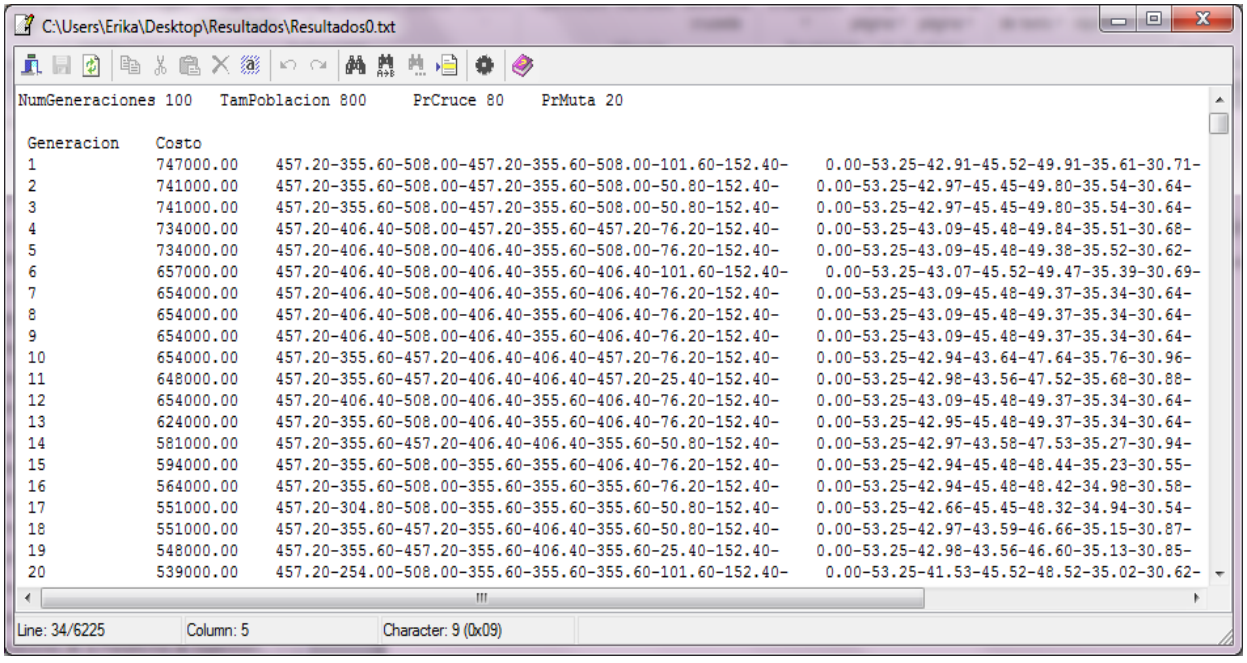
```
-- Obtenemos el mayor de cada Bloque
Select  T.NomArchivo,
        Bloque      = 'BLOQUE ' + T.Bloque,
        Generacion  = T.IdGeneracion,
        Maximo      = T.Valor
From    (   Select  B.NomArchivo,
                    Bloque          = B.Bloque,
                    IdGeneracion    = Max(B.IdGeneracion)
            From    (
                        Select  NomArchivo,
                                Bloque      = Bloque,
                                Valor       = Max(Valor)
                        From    #tInformacionArchivo
                        Group by NomArchivo, Bloque
                    ) As A
                    Inner Join #tInformacionArchivo as B
                    On  B.Bloque            = A.Bloque
                        And B.Valor         = A.Valor
                        And B.NomArchivo    = A.NomArchivo
            Group   By B.NomArchivo, B.Bloque
        )   As O
        Inner Join  #tInformacionArchivo as T
        On T.Bloque             = O.Bloque
            And T.IdGeneracion  = O.IdGeneracion
            And T.NomArchivo    = O.NomArchivo
Order by T.NomArchivo, Convert(Int,T.Bloque) Asc

-- Bloques mayores
Select  * From #tInformacionArchivoMax

-- Promedio x Bloque
Select  NomArchivo,
        Bloque      = 'BLOQUE ' + Bloque,
        --Generacion    = IdGeneracion,
        Promedio    = AVG(Valor)
From    #tInformacionArchivo
Group   By NomArchivo, Bloque
Order by NomArchivo, Convert(Int,Bloque) Asc, Promedio Asc

-- Promedio general
Select  NomArchivo, Promedio    = AVG(Valor)
From    #tInformacionArchivo
Group   By NomArchivo
Order   By NomArchivo

-- Promedio por generación
Select  NomArchivo, IdGeneracion, Promedio = AVG(Valor)
From    #tInformacionArchivo
Group   By NomArchivo, IdGeneracion
Order   By NomArchivo, IdGeneracion

Drop    Table #tInformacionArchivo,#tInformacionArchivoMax, #tInformacionArchivoMin
```

Estructura de los Archivos de Salida del Algoritmo Evolutivo.

Estructura del Archivo Aptitudes



Estructura del archivo Resultados



Apéndice III. Infraestructura Utilizada

En este apartado se presentan las especificaciones detalladas de la infraestructura de hardware y software utilizadas para la implementación del algoritmo evolutivo paralelo PEA-WDND, desarrollado en este trabajo.

Infraestructura Hardware y Software del Clúster Cuexcomate

ELEMENTO	HARDWARE	SOFTWARE
Comunicaciones	Switch 3COM 24/10/100/1000 Switch Infiniband Mellanox de 18 puertos de 40 Gb/s	NODO MAESTRO Y DE CÓMPUTO • S.O. Centos 5.5 • Torque 2.3 con Maui 3.2 • Ganglia • Cluster Instalation and Administration (CIA) • Intel Cluster Toolkit compiler edition for linux academia
Nodo maestro CPU Total 12 cores Total 24 GB RAM Total 12 TB HD	1 Motherboard: • 2 procesadores Intel Xeon Six Core a 3.06 GHz, 12 MB cache. • 2 HD Enterprise, 7200 RPM de 500GB (para S.O.). • 6 HD Enterprise, 7200 RPM, 12 TB en total. • 6 modulos RAM de 4GB 1333MHZ DDR3. Total de 24 GB RAM. • 1 tarjeta Infiniband 40Gb/s	NODO GPU • S.O. Centos 5.5 • Drivers propietarios Nvidia • Compiladores GNU-Linux GCC • Compilador Nvidia-CUDA • Depuradores: Depuración codigos paralelos en híbrido CPU-GPU. GNU. Front-End gráfico para todo lenguaje. Nativo para GPU de Nvidia. • APIs para programación paralela: OpenMP, MPI, OpenCL-Nvidia. • Bibliotecas cómputo numérico: para desarrollo de aplicaciones CPU-GPU. Para cómputo científico en varios lenguajes, en sus versiones serial y paralela: BLAS, LAPACK, Científica GNU para C y C++, ATLAS, FFW, CUBLAS, CUFFT, CURAND,
Nodos de procesamiento CPU 01 al 04 Total 48 cores Total 96 GB RAM Total 2 TB HD	1 Motherboard: • 2 procesadores Intel Xeon Six Core a 3.06 GHz, 12 MB cache. • 1 HD Enterprise, 7200 RPM de 500GB. • 6 modulos RAM de 4GB 1333MHZ DDR3. Total de 24 GB RAM. 1 tarjeta Infiniband 40Gb/s	
Nodo de procesamiento GPU	1 Motherboard: • 1 procesador Intel Xeon Six Core a 3.06 GHz, 12 MB	

05 Total 896 cores Total 36 GB RAM Total 1 TB HD	cache. • 2 HD Enterprise, 7200 RPM de 500GB. • 9 modulos RAM de 4GB 1333MHZ DDR3. Total de 36 GB RAM. • 2 tarjetas NVIDIA TESLAC2070, arquitectura Fermi, con 6 GB RAM DDR5 c/u. 448 cores c/u. • DVD/RW • Lector de memoria 1 tarjeta Infiniband 40Gb/s	CURPARSE. • Editor Vim • IDE Eclipse Helios • Herramientas de monitoreo CUDA: CUDA profiler, CUDA memcheck, nvidia-smi, cudafe, cudafe++. • OpenMPI 1.2.8 • MPICH2-1.0.8 • OpenVPN

Infraestructura Hardware y Software del Clúster de pruebas CIICAp 2.
Ubicación: Laboratorio de Optimización, CIICAp, UAEM, Cuernavaca, Morelos.

ELEMENTO	HARDWARE	SOFTWARE
Comunicaciones	Switch Cisco C2960 24/10/100 Cableado interno nivel 5	
Nodo maestro	Pentium 4, 2793 MHz 512 MB Memoria 80 GB disco duro 2 tarjetas 10/100 Mb/s	S.O. Red Hat Enterprise Linux 4 Compilador gcc versión 3.4.3 OpenMPI 1.2.8 MPICH2-1.0.8 Ganglia 3.0.6 NIS ypserv-2.13-5 NFS nfs-utils-1.0.6-46 OpenVPN
Nodos de procesamiento 01 al 18	Intel® Celeron® Dual Core, 2000MHz, RAM 2GB RAM, 160GB disco duro, tarjeta 10/100 Mb/s	

Infraestructura Hardware y Software del Clúster Texcal.
Ubicación: Laboratorio de Cómputo, UPEMOR, Jiutepec, Morelos, México.

ELEMENTO	HARDWARE	SOFTWARE
Comunicaciones	Switch 3COM 24/10/100/1000 Switch Infiniband Mellanox de 18 puertos de 40 Gb/s	NODO MAESTRO Y DE CÓMPUTO • S.O. Centos 5.5 • Torque 2.3 con Maui 3.2 • Ganglia • Cluster Instalation and Administration (CIA) • Intel Cluster Toolkit compiler edition for linux academia
CPU Total 12 cores Total 24 GB RAM Total 12 TB HD	Motherboard: • 2 procesadores Intel Xeon Six Core a 3.06 GHz, 12 MB cache. • 2 HD Enterprise, 7200 RPM de 500GB (para S.O.).	

	• 6 HD Enterprise, 7200 RPM, 12 TB en total. • 6 módulos RAM de 4GB 1333MHZ DDR3. Total de 24 GB RAM. • 1 tarjeta Infiniband 40Gb/s	NODO GPU • S.O. Centos 5.5 • Drivers propietarios Nvidia • Compiladores GNU-Linux GCC • Compilador Nvidia-CUDA • Depuradores: Depuración códigos paralelos en híbrido CPU-GPU. GNU. Front-End gráfico para todo lenguaje. Nativo para GPU de Nvidia. • APIs para programación paralela: OpenMP, MPI, OpenCL-Nvidia. • Bibliotecas cómputo numérico: para desarrollo de aplicaciones CPU-GPU. Para cómputo científico en varios lenguajes, en sus versiones serial y paralela: BLAS, LAPACK, Científica GNU para C y C++, ATLAS, FFW, CUBLAS, CUFFT, CURAND, CURPARSE. • Editor Vim • IDE Eclipse Helios • Herramientas de monitoreo CUDA: CUDA profiler, CUDA memcheck, nvidia-smi, cudafe, cudafe++. • OpenMPI 1.2.8 • MPICH2-1.0.8 • OpenVPN
CPU 01 al 04 Total 48 cores Total 96 GB RAM Total 2 TB HD	Motherboard: • 2 procesadores Intel Xeon Six Core a 3.06 GHz, 12 MB cache. • 1 HD Enterprise, 7200 RPM de 500GB. • 6 modulos RAM de 4GB 1333MHZ DDR3. Total de 24 GB RAM. 1 tarjeta Infiniband 40Gb/s	
GPU 05 Total 896 cores Total 36 GB RAM Total 1 TB HD	Motherboard: • 1 procesador Intel Xeon Six Core a 3.06 GHz, 12 MB cache. • 2 HD Enterprise, 7200 RPM de 500GB. • 9 módulos RAM de 4GB 1333MHZ DDR3. Total de 36 GB RAM. • 2 tarjetas NVIDIA TESLAC2070, arquitectura Fermi, con 6 GB RAM DDR5 c/u. 448 cores c/u. • DVD/RW • Lector de memoria 1 tarjeta Infiniband 40Gb/s	

Apéndice IV. Lista de Publicaciones

En este apartado se presenta un listado de las publicaciones desarrolladas durante el desarrollo de este trabajo doctoral. Algunas de estas han sido publicadas en congresos y otras de ellas en revistas especializadas.

1. M.A. Cruz-Chávez and E.Y. Avila-Melgar, Scheduling in Water Distribution Networks; International congress of Computation in Optimization and Software 2009. ISBN 978-607-00-1970-8. Cuernavaca Morelos, México.

2. Marco Antonio Cruz-Chávez, Erika Yesenia Ávila-Melgar, Fredy Juárez Pérez, Wiston G. Torres-Sánchez, Empirical Transformation of Job Shop Scheduling Problem to the Hydraulic Networks Problem in a Water Distribution System, Electronics, Robotics and Automotive Mechanics Conference, CERMA2009, IEEE-Computer Society, ISBN 978-0-7695-3799-3, pp 76-81, September 22 - 25, México, 2009.

3. Marco Antonio Cruz-Chávez, Abelardo Rodriguez-León, Erika Yesenia ávila-Melgar, Fredy Juárez-Pérez, José Crispín Zavala-Díaz, Rafael Rivera-López, Parallel Hybrid Evolutionary Algorithm in a Grid Environment for the Job Shop Scheduling Problem, Proceedings of the Second EELA-2 Conference, CIEMAT, ISBN 978-84-7834-627-1, pp 227 - 234, November 25-27, Choroní, 2009

4. R. Baños, C.M. Fonseca, C.Gil, A.Márquez, E.Y. Ávila-Melgar, F.G. Montoya, Design and Evaluation of Evolutionary Operators for Water Distribution Network Optimization, International Conference on Metaheuristics and Nature Inspired Computing META´10, Djerba, Tunisia, Octubre 2010. Djerba Island, Tunisia.

5. Marco Antonio Cruz-Chávez, Abelardo Rodríguez-León, Erika Yesenia Ávila-Melgar, Fredy Juárez-Pérez, Martín H. Cruz-Rosales, Rafael Rivera-López, Genetic-Annealing Algorithm in Grid Environment for Scheduling Problems, Communications in Computer and Information Science: Security-Enriched Urban Computing and Smart Grid, Springer Verlag Pub., Berlin Heidelberg, ISSN: 1865-0929, Vol. 78, pp.1-9, 2010.

6. Marco Antonio Cruz-Chávez, E.Y. Avila-Melgar, Sergio A. Serna Barquera, Fredy Juárez-Pérez, General Methodology for Converting a Sequential Evolutionary Algorithm into Parallel Algorithm with MPI to Water Design Networks, Electronics, Robotics and Automotive Mechanics Conference, CERMA2010, IEEE-Computer Society, ISBN 978-0-7695-4204-1, pp 149 - 154, September 28 - October 1, México, 2010.

7. Marco Antonio Cruz-Chávez, Abelardo Rodríguez-León, E.Y. Ávila-Melgar, Fredy Juárez-Pérez, Martín H. Cruz-Rosales, Rafael Rivera-López, Gridification of Genetic Algorithm with Reduced Communication for the Job Shop Scheduling Problem, International Journal of Grid and Distributed Computing, Science and Engineering Research Support soCiety, Australia, ISSN: 2005-4262, Vol. 3, No. 3, pp. 13-28, 2010 (FREE PAPER).

8. Marco Antonio Cruz-Chávez, et al., Solutions Space Analysis for the Combined Mathematical Model (Linear and Nonlinear) of the Water Distribution Network Design Problem, Lecture Notes in Artificial Intelligence, Springer Verlag Pub., Berlin Heidelberg, ISSN: 0302-9743, Vol., pp., 2014

Referencias Bibliográficas

[Abebe, 1998a] Abebe A. J. and Solomatine D.P., Application of global optimization to the design of pipe networks, Proceedings of the International Conference on Hydroinformatics, 989-996, A.A.Balkema, Brookfield, 1998.

[Abebe, 1998b] Solomatine D.P. Genetic and other global optimization algorithms - comparison and use in model calibration. Proc. Intern Conf. Hydroinformatics-98, Balkema, Rotterdam, 1998.

[Abebe, 1998c] Abebe A. J. and Solomatine D.P. Two Strategies of Adaptative Cluster Covering with Descent and Their Comparison to Other Algorithms, Journal of Global Optimization 14:55-79, 1998.

[Ada, 2010] Ada Gavrilovska, Attaining High Performance Communications: A Vertical Approach, ed. Ada Gavrilovska, 2010

[AGU, 2000] Web de American Geophysical Union, http://www.agu.org/

[Alba, 2002] Alba Enrique, Parallel evolutionary algorithms can achieve superlineal performance. Information Processing Letters, April 2002

[Alba, 2005] Alba E, Luque G (2005) Measuring the performance of parallel metaheuristics. In: Alba E (ed) Parallel metaheuristics - a new class of algorithms, pp 43-62. Wiley Series on Parallel and Distributed Computing, Hoboken, New Jersey

[Alba, 2011] Alba Enrique , Parallel Genetic Algorithms: Theory and Real World Applications, Springer 2011

[Alperovits, 1977] Alperovits E., Shamir U. Design of Optimal Water Distribution Systems, Water Resources Research,13(6),885-900,1977

[Amor, 2005] Amor, B.H., Rettinger, A Intelligent Exploration for genetic Algorithms: using self-organizing maps in evolutionary computation. Proceedings of 2005 conference on Genetic and evolutionary computation, 2005.

[Amor,2008] Ben Amor, Intelligent Exploration for Genetic Algorithms. Using Self-Organizing Maps in Evolutionary Computation, VDM Verlag, 2008.

[Back, 1997] Back T., Fogeld. B., and Michalewicz Z. Handbook of Evolutionary Computation. Oxford University Press, 1997.

[Baños, 2006] R. Baños Navarro, Meta heurísticas Híbridas para Optimización Mono-objetivo y Multi-objetivo, Tesis Doctoral, Almería, España, Diciembre 2006. Disponible en: www.ace.ual.es/~rbanos/CV.html, última fecha de acceso 12 Mayo de 2009.

[Baños, 2007] R. Baños Navarro, Implementation of scatter search for multi-objective optimization: a comparative study, 15 November 2007 © Springer Science+Business Media, LLC 2007

[Baños, 2010] Baños R., Gil C., Reca J., Martinez J.: "Meta-heuristics Applied to the Optimal Design of Water Distribution Systems". In Computational Optimization: New Research Developments. Ed. Nova Publishers. pp.145-168. 2010. ISBN: 978-1-60692-671-0.

[Baños, 2010b] Baños R., Fonseca C.M., C.Gil, Márquez A., Ávila-Melgar E.Y., Montoya F.G., Design and Evaluation of Evolutionary Operators for Water Distribution Network Optimisation, . International Conference on Metaheuristics and Nature Inspired Computing META´10, Djerba, Tunisia, October 27-31, 2010.

[Becker, 2010] Becker Brett, High Level Data Partitioning For Parallel Computing on Heterogeneous, 2010.

[Bellatreche, 2011] Bellatreche Ladjel, Mota Filipe, Model and Data Engineering: First International Conference, Medi 2011.

[Berge, 1973] C. Berge. Graphs and hypergraphs. North-Holland, 1973.

[Berman, 2003] Berman Fran, Geoffrey, Grid Computing: Making the Global Infrastructure a Reality, 2003

[Bhave,1991] Bhave, P.R.Analysis of Flow in Water Distribution Networks, Ed. Technomic Publishing Company, Lancaster, Pennsylvania (EEUU), 1991.

[Blazewicz, 2000] Blazewicz, J. Handbook on Parallel and Distributed Processing in International Hanbooks on Information Systems, Springer 2000.

[Bondy, 1976] J. A. Bondy and U. S. R. Murty. Graph theory with applications. University Press, Belfast, 1976.

[Cantu, 1997] Cantu-Paz, E., A Survey of Parallel Genetic Algorithms, Technical Report IlliGAL 97003, University of Illinois at Urbana-Champaign, 1997.

[Cantu, 1999] Cantú-Paz, E. Topologies, migration rates, and multi-population

parallel genetic algorithms, 1999.

[Cantu, 2000] Cantú-Paz, E.,Efficient and accurate parallel genetic algorithms. Springer, 2000.

[Chakrabarti, 2007] Chakrabarti Anirban, Grid Computing Security.

[Chandra, 2010] Chandra Pramod, Bhatt P. An Introduction to Operating Systems: Concepts and Practice, 2010,

[Chistofides, 1975] N. Chistofides. Graph Theory. An algorithmic approach. Academic Press, New York, 1975.

[Coello, 2010] Coello Coello Carlos A, B, Gary, Lamont, A. David, Veldhuizen Van, Evolutionary Algorithms for Solving Multi-Objective Problems, 2010.

[Cook, 1971] Cook S. A, The complexity of theorem proving procedures Proc. 3rd ACM symposium on theory of computing. Shaker Heights, Ohi. 151-158, 1971.

[Cormen, 2001] Cormen, T., Leiserson, Ch., Rivest, R., Introduction to Algorithms. Mc.Graw Hill, 2001

[Crainic, 2003] Crainic T.G. Toulouse M., Parallel Strategies for Meta-heuristics, F. Glover, G.A. Kochenberber, (Eds.). Handbook of Metaheuristics. Kluwer Academics, 2003.

[Crainic, 2010] Crainic, T. G. and Toulouse, M. Parallel meta-heuristics. In Gendreau, M. and Potvin, J.-Y., editors, Handbook of Metaheuristics, pages 497–541. Springer, 2010.

[Cruz, 2009a] M.A. Cruz-Chávez and E.Y. Avila-Melgar, Scheduling in Water

Distribution Networks; International congress of Computation in Optimization and Software 2009. ISBN 978-607-00-1970-8. Cuernavaca Morelos, México.

[Cruz, 2009b] M.A. Cruz-Chávez and E.Y. Avila-Melgar. Empirical Transformation of Job Shop Scheduling to the Hydraulic Networks in a Water Distribution System; The Electronics, Robotics and Automotive Mechanics Conference, 2009. ISBN 978-0-7695-3799-3. Xochitepec Morelos, México.

[Cruz, 2009c] M.A. Cruz-Chávez and E.Y. Avila-Melgar, Parallel Hybrid Evolutionary Algorithm in a Grid Environment for the Job Shop Scheduling Problem. International congress of E-science Grid facility for Europe and Latin America. EELA-2 Conference, Choroni, Venezuela.

[Cruz, 2010a] Marco Antonio Cruz-Chávez, Abelardo Rodríguez-León, Erika Yesenia Ávila-Melgar, Fredy Juárez-Pérez, Martín H. Cruz-Rosales, Rafael Rivera-López, Gridification of Genetic Algorithm with Reduced Communication for the Job Shop Scheduling Problem, International Journal of Grid and Distributed Computing, Science and Engineering Research Support soCiety, Australia, ISSN: 2005-4262, Vol. 3, No. 3, pp. 13-28, 2010.

[Cruz, 2010b] Marco Antonio Cruz-Chávez, Abelardo Rodríguez-León, Erika Yesenia Ávila-Melgar, Fredy Juárez-Pérez, Martín H. Cruz-Rosales, Rafael Rivera-López, Genetic-Annealing Algorithm in Grid Environment for Scheduling Problems, Communications in Computer and Information Science: Security-Enriched Urban Computing and Smart Grid, Springer Verlag Pub., Berlin Heidelberg, ISSN: 1865-0929, Vol. 78, pp.1-9, 2010.

[Cruz, 2010c] M.A. Cruz-Chávez and E.Y. Avila-Melgar, General Methodology for Converting a Sequential Evolutionary Algorithm into Parallel Algorithm with MPI as Applied to Water Design Networks; The Electronics, Robotics and Automotive Mechanics Conference, 2010. Xochitepec Morelos, México.

[Cruz, 2012b] M.A. Cruz-Chávez , F. Juárez-Pérez y P. Moreno Bernal, MiniGrid Morelos , inter-agency synergy, Hypatia, Revista de Divulgación Científico-Tecnológica del Consejo de Ciencia y Tecnología del Estado de Morelos , Editor Responsable: Silvia Patricia Pérez Sabino, No. 40. Pág.32-33, Enero/Marzo de 2012.

[Cruz, 2012] Logistics Management and Optimization through Hybrid Artificial Intelligence Systems, Cruz-Chávez M. A, et al., Chapter: Marco Antonio Cruz-Chávez et al., Grid Platform Applied to the Vehicle Routing Problem with Time Windows for the Distribution of Products, ISBN13: 9781466602977, IGI Global, Editor Carlos Alberto Ochoa Ortiz Zezzatti., 422 pages, march 2012.

[Cruz, 2014] Marco Antonio Cruz-Chávez, et al., Solutions Space Analysis for the Combined Mathematical Model (Linear and Nonlinear) of the Water Distribution Network Design Problem, Lecture Notes in Artificial Intelligence, Springer Verlag Pub., Berlin Heidelberg, ISSN: 0302-9743, Vol., pp., 2014

[Cruz, 2014b] Marco Antonio Cruz-Chávez, Neighborhood Generation Mechanism Applied in Simulated Annealing to Job Shop Scheduling Problems, International Journal of Systems Science, Taylor & Francis, ISSN: 0020-7721, DOI:10.1080/00207721.2013.876679, pp.1-13, 2014.

[Cunha, 1999] Cunha, M.C. and Sousa J. Water distribution network design optimization: Simmulated Annealing approach, J. Water Resources Planning Management, 125 (4), 215-221.

[Dandy, 1996] Dandy GC, Simpson AR, Murphy LJ (1996) An improved genetic algorithm for pipe network optimization. Water Resources 32(2):449–458. doi:10.1029/

[Darwin, 1859] Darwin, C. On the origins of species by means of natural selection. London: Murray, 1859.

[Darwin, 1872] Darwin. On the Origin of Species by means of Natural Selection, or the Preservation of Favored Races in the Struggle for Life. John Murray, London, 1872.

[Diestel, 2010] Reinhard Diestel. Graph Theory. GTM 173, 4th edition 2010. Corrected reprint 2012. Springer-Verlag, Heidelberg Graduate Texts in Mathematics, Volume 173.

[Diego-mas, 2006] Diego-más José Antonio, Optimización de la Distribución en planta de Instalaciones Industriales mediante algoritmos genéticos, Tesis Doctoral, Valencia, España, Enero 2006.

[De-Jong, 1990] K. De Jong, K.A. and W. M. Spears. “An analysis of the interacting roles of population size and crossover in genetic algorithms”, Springer Verlag, 1990.

[Duarte, 2006] Duarte A., Pantrigo Fernandez J.J., Gallego Carrillo Michael, Heurísticas, Tesis Doctoral, 2006.

[Epanet, 2015] http://www.epa.gov/nrmrl/wswrd/dw/Epanet.html. Fecha de ultimo acceso, Marzo 2015.

[Eiben, 1999] A.E. Eiben, R. Hinterding, and Z. Michalewicz, Parameter control in evolutionary algorithms, IEEE Transactions on Evolutionary Computation, 3(2):124-141, 1999.

[Eiben, 2014] G. Karafotias, M. Hoogendoorn, and A.E. Eiben, Parameter Control in Evolutionary Algorithms:Trends and Challenges, IEEE Transactions on Evolutionary Computation, 2014. DOI: 10.1109/TEVC.2014.2308294.

[Eusuff, 2003] Eusuff, M. M., and K. E. Lansey (2003), Optimization of water distribution network design using the shuffled frog leaping algorithm, J. Water Resour. Plann. Manage., 129(3), 210– 225 http://www.epa.gov/nrmrl/wswrd/dw/Epanet.html.

[Fogel, 1965] L. Fogel, A. Owens, M. Walsh. Artificial Intelligence through a simulation of evolution. Biophysics and Cybernetic Systems: Proceedings of the 2nd Cibernetic Sciences Symposium, 131-155, 1965.

[Foster, 2002a] What is the Grid? A three checklist. Grid Today, 2002.

[Foster, 2002b] Foster I., Kesselman C., Tuecke S. The Anatomy of the Grid:Enabling Scalable Virtual Organizations. International Journal of Supercomputing Applications, 2002.

[Fujiwara, 1987] Fujiwara O., Jenchimahakonn B. A Modified Linear Programming Gradient Method for Optimal Design of Looped Water Distribution Networks, Water Resources Research, 23(6), 997/982, 1987.

[GCC, 2013] Web de compilador GCC http://gcc.gnu.org/, 2013

[Geem, 2001] Geem, Z.W., Kim J.H. and Loganathan G.V. A new heuristic optimization algorithm: Harmony search, Simulation, 76(2), 60-68, 2001.

[Ghanea, 2003] Ghanea R., Hercock, Applied Evolutionary Algorithms, Springer Verlag, New York, 2003

[Gil, 2011] Gil C., Baños R., Ortega J., Márquez A.L., Fernández A., Montoya M.G. “Ant Colony Optimization for Water Distribution Network Design: A Comparative Study” IWANN 2011. Malaga, Spain. June 8-10, 2011.

[Glover, 1986] F. Glover. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, 13:533 549, 1986.

[Goedecker, 2001] Goedecker Stefan ,Adolfy, Performance Optimization of Numerically Intensive Codes, Performance Optimization of Numerically Intensive Codes, 2001.

[Goldberg, 1989] Goldberg, D. E. Genetic Algorithms in search, optimization and machine learning. Addison-Wesley Publishing Co., Reading Mass, 1989.

[Goulter, 1986] Goulter I. C.,Lussier B.M., and Morgan D. R., Implications of head loss path choice in the optimization of water distribution

networks, Water Resources Research, 22 (5), 819-882, 1986

[Gupta, 1993] Gupta, I., Bassin, J.K., Gupta A., Khanna P., Optimization of Water Distribution System, Environmental Software 8, 101-113, 1993.

[Gutin, 2004] G. Gutin and A. Punnen. The traveling salesman problem and its variations. Kluwer Academic Publishers, 2004.

[Hinterding, 1995] Hinterding R., Gielewski H., Peachey T.C., The nature of mutation in genetic algorithms, Morgan Kaufmann, San Mateo, 1995.

[Holland, 1975] J. H. Holland, Adaptation in natural and artificial systems. MIT Press, Cambridge, Mass, 1975.

[Hoare, 1962] Hoare,C.A.R.; Quicksort. Computer Journal 5(1),10-15,1962.

[Horn, 1994] J. Horn, N. Nafpliotis, D.E. Goldberg, “A niched pareto genetic algorithm for multiobjective optimization”, Proceedings of the IEEE Conference on Evolutionary Computation, Orlando (USA), June 1994, Vol. 1, pp. 82-87

[Kelly, 1996] J.P. Kelly and I.H. Osman. Meta-Heuristic: Theory and Applications. Kluwer Academic Publisher, 1996.

[Kessler, 1989] Kessler A. and Shamir U. Analysis of the linear programming gradient method for optimal design of water supply networks, Water Resources Research, 25 (7), 1469-1480, 1989.

[Koza, 1992] Koza J.R. Genetic Programming: on the programming of computers by means of natural selection. MIT Press, 1992.

[Landman, 2008] Landman Joe. MPI in thirthy minutes

[Loganathan, 1990] Loganathan, V.G., Sherali H.D. and Shah M.P. A two-phase network design heuristic for minimum cost water distribution system under a reliability constraint, Eng.Optim., 15 (4),311-336,1990.

[Loganathan, 1995] Loganathan, V.G., Greene J. J., and Ahn T. J., Design heuristic for globally minimum cost water distribution systems, Water Resources Research,121(2), 182-192, 1995.

[López-Ibáñez, 2008] López-Ibáñez Manuel, T. Devi Prasad, and Ben Paechter., Parallel Optimisation of Pump Schedules with a Thread-Safe Variant of Epanet Toolkit, Proceedings of the 10th Annual Water Distribution Systems Analysis Conference WDSA2008, Van Zyl, J.E., Ilemobade, A.A., Jacobs, H.E. (eds.), August 17-20, 2008, Kruger National Park, South Africa.

[López-Ibáñez, 2011] López-Ibáñez Manuel, T. Devi Prasad, and Ben Paechter. Representations and Evolutionary Operators for the Scheduling of Pump Operations in Water Distribution Networks. Evolutionary Computation, 19(3):429–467, 2011.

[Lorente, 2012] Lorente D. GTriguero., I., Gil C., Espín A.. “Evolutionary Algorithms for the Design of Grid-Connected PV-systems". Expert Systems with Applications, Vol. 39, N. 9, pp. 8086-8094. 2012.

[Melián,2003] Melián, Belén. Pérez, José A. et al."Metaheurísticas: una visión global". Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. N.19 pp. 7-28 ISSN: 1137-3601. © AEPIA(2003).

[Michalewicz, 1992] Michalewicz, Z. Genetic algorithms + data structures = evolution programs. Springer-Verlag, New York, 1992.

[Montesinos, 1999] Montesinos P. Garcia-Guzman A., Ayuso J.L. Water Distribution Network optimization using a modified genetic algorithm, Water Resources Research, 35(11), 3467-3473, 1999.

[MPI, 2013] Web del estándar MPI http://www.mcs.anl.gov/research/projects/mpi/, 2013

[MPI2, 2013] Web del estándar MPI2 http://www.mcs.anl.gov/research/projects/mpi/mpi2-forum/mpi2.html , 2013

[MPICH, 2013] Web del estándar MPICH2 http://www.mcs.anl.gov/research/projects/mpich2/, 2013. Última fecha de acceso Noviembre de 2013.

[Nazif, 2009] Nazif Sara, Karamouz Mohammad, Tabesh Massoud, Moridi Ali, Pressure Management Model for Urban Water Distribution Networks, Water Resour Manage, DOI 10.1007/s11269-009-9454-x. Springer Science+Business Media B.V. 2009

[OpenMPI, 2015] Web de OpenMPI http://www.open-mpi.org/, 2015. Última fecha de acceso Marzo de 2015.

[Ostfeld, 2005] Ostfeld Avi, J. Water Distribution Systems Connectivity Analysis. Water Resour. Plng. and Mgmt. 131, 58 (2005),DOI:10.1061/(ASCE)0733-9496(2005)131:1(58)

Papadimitriou, 1998] C.H. Papadimitriou and K. Steiglitz. Combinatorial Optimization. Dover Publications,

[Quindry, 1979] Quindry, G.E., E.D. Brill, J.C. Liebman, and A.R. Robinson.

Comment on “Design of Optimal Water Distribution Systems, Water Resources Research” by Alperovits E., Shamir U. Water Resources Research,15(6), 1651-1654

[Rajkumar, 1999] Rajkumar Buyya, High Performance Cluster Computing: Architectures

[Rauber, 2010] Rauber Thomas, Rünger Gudula, Parallel Programming: For Multicore and Cluster Systems

[Reca, 2008] Reca J., Martinez J., Baños R., Gil C. Optimal Design of Gravity-Fed Looped Water Distribution Networks Considering the Resilience Index Water Resources Planning and Management, ASCE 2008.

[Reca, 2006] Reca J., Martinez J. Genetic Algorithms for the design of looped irrigation water distribution networks, Water Resources Research, Vol. 42, Spain, 2006.

[Rechenberg, 1973] I. Rechenberg. Evolutionstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. Frommann-Holzboog, 1973.

[Rodriguez, 2010] Abelardo Rodriguez-León, Marco Antonio Cruz-Chávez, Rafael Rivera-López, Erika Yesenia Ávila-Melgar, Fredy Juárez-Pérez, Ocotlán Díaz-Parra, A Communication Scheme for an Experimental Grid in the Resolution of VRPTW using an Evolutionary Algorithm, Electronics, Robotics and Automotive Mechanics Conference, CERMA2010, IEEE-Computer Society, ISBN 978-0-7695-4204-1, pp 108 - 113, September 28 - October 1, México, 2010.

[Ropo, 2009] Ropo Matti, Westerholm Jan, Dongarra Jack Recent Advances

in Parallel Virtual Machine and Message Passing Interface

[Rossman, 2000] EPANET 2 User's manual. US Environmental Protection Agency, Cincinnati, OH

[Rucínski, 2010] Rucínski, M., Izzo, D., and Biscani, F. (2010). On the impact of the migration topology on the island model. Parallel Computing, 36(10):555–571.

[Savic ,1997] Savic D.A., Walters G.A. Genetic Algorithms for Least-cost Design of Water Distribution Networks, Journal of Water Resources Planning and Management 123(2), 67-77,1997.

[Sedgewick, 1984] Sedgewick Robert, Algorithms, Addison Wesley ISBN O-201-06672-6, 1984.

[Sherali ,1998] Sherali, Totlani R., Loganathan G.V. Enhanced Lower Bounds for the Global Optimization of Water Distribution Networks, Water Resources Research 34 (7), 1831-1841, 1998.

[SQL, 2005] http://microsoft-sql-server.software.informer.com/9.0/. Última fecha de acceso: 23/05/2014

[Skolicki,2005a] Skolicki, Z. An analysis of island models in evolutionary computation. In Proc. of the 2005 Workshops on Genetic and Evolutionary Computation, pages 386–389. ACM, 2005.

[Skolicki,2005b] Skolicki, Z. and De Jong, K. The influence of migration sizes and intervals on island models. In Proc. of the 2005 Conference on Genetic and Evolutionary Computation, pages1295–1302. ACM, 2005.

[Stinson ,1987] D. R. Stinson; An Introduction to the Design and Analysis of

Algorithms; Second Edition (Revised); Winnipeg, Manitoba, Canada. 1987.

[Syswerda,1989] Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms. Proceedings of the Third International Conference on Genetic Algorithms, ed. J. D. Schaffer, San Mateo, CA: Morgan Kaufmann, 2-8.

[Talbi,2002] Talbi E (2002) A taxonomy of hybrid metaheuristics. J Heuristics 8(5):541–564

[Talbi,2009] Talbi, E.-G. Metaheuristics: From design to implementation. John Wiley & Sons, 2009.

[Thakur, 2005a] Thakur Rajeev, Rabenseifner Rolf, Gropp William, Optimization of Collective Communication Operations in MPICH International Journal of High Performance Computing Applications Spring 2005.

[Thakur, 2005b] Thakur Rajeev, Rabenseifner Rolf, Gropp William, Optimizing the Synchronization Operations in MPI One-Sided Communication.

[Tsaregorodtsev, 2009] Tsaregorodtsev, Hammar Vannesa, DIRAC experience on EELA-2, Proceedings of the Second EELA-2 Conference, CIEMAT, ISBN 978-84-7834-627-1, pp 59-68, November 25-27, Choroni, 2009.

[Vairavamoorthy, 2000] Vairavamoorthy, K., and M. Ali, Optimal design of water distribution systems using genetic algorithm, Comput. Aided Civ. Infrastruct Eng., 15(2), 374–382, 2000.

[Varma,1997] Varma, K.V.K., Narasimhan S., and Bhallamudi S.M. Optimal design of water distribution systems using NLP method, J.

Environ.Eng,123 (4), 381-388.

[Voss,1999] S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors. Metaheuristic: Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers, Boston, 1999.

[Wilkinson, 2008] Wilkinson Barry, Grid Computing: Techniques and Applications.

[Zanakis, 1981] H. Zanakis, J.R. Stelios, and A. Evans. Heuristic optimization: Why, when and how to use it. Interfaces, 5(11):84-90, 1981.

[Zhang, 2012] Zhang Tianbiao, Instrumentation, Measurement, Circuits and Systems