



UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Herramienta para la optimización de redes de distribución de agua potable

GABRIEL GONZALO ALEXANDER SANHUEZA FUENTES

Profesor Guía: JIMMY GUTIÉRREZ BAHAMONDES

Profesor Co-guía: DANIEL MORA MELIA

Memoria para optar al título de
Ingeniero Civil en Computación

Curicó – Chile
Julio, 2020



UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Herramienta para la optimización de redes de distribución de agua potable

GABRIEL GONZALO ALEXANDER SANHUEZA FUENTES

Profesor Guía: JIMMY GUTIÉRREZ BAHAMONDES

Profesor Co-guía: DANIEL MORA MELIA

Profesor Informante: PROFESOR INFORMANTE 1

Profesor Informante: PROFESOR INFORMANTE 2

Memoria para optar al título de
Ingeniero Civil en Computación

El presente documento fue calificado con nota: _____

Curicó – Chile

Julio, 2020

Dedicado a mis padres, hermanos y amigos.

AGRADECIMIENTOS

Al terminar este proceso quiero agradecer en primer lugar a Dios y a mi familia, a mi madre Doris Fuentes, mi padre Julian Sanhueza, mis hermanos Bastian Sanhueza y el pequeño Joaquín por ser mis mayores pilares y apoyo.

También agradecer a los profesores Jimmy Gutiérrez Bahamondes y Daniel Mora Melia por darme la oportunidad de participar en este proyecto y por su cooperación en el desarrollo y finalización de éste.

A mis amigos, especialmente a Ariel Cornejo, Diego Iturriaga, Diego Matus, Felipe Milla, José Nuñez, Benjamín Pino y Roberto Ureta y Maximiliano Viveros por darme ánimos para continuar adelante y no rendirme en el camino.

Finalmente, agradecer al proyecto Fondecyt Regular N° 1180660 “Optimization of real-world water distribution systems and hydraulic elements using computational fluid dynamics (cfd) and evolutionary algorithms.”, 2018.

TABLA DE CONTENIDOS

	página
Dedicatoria	I
Agradecimientos	II
Tabla de Contenidos	III
Índice de Figuras	VI
Índice de Tablas	VIII
Resumen	IX
1. Introducción	10
1.1. Contexto del proyecto	10
1.2. Presentación del problema	11
1.3. Objetivos	11
1.4. Propuesta de solución	12
1.5. Alcances	13
2. Marco Teórico	15
2.1. Metodología de desarrollo	15
2.2. Metodología de evaluación	16
2.3. Herramientas para el desarrollo del software	17
2.3.1. Lenguaje de programación Java	17
2.3.2. Patrones de diseño	19
2.3.3. Epanet Programming Toolkit	20
2.4. Hidráulica húrbana	21
2.4.1. Conceptos básicos de hidráulica	21
2.4.2. Red de distribución de agua	21
2.5. Optimización	23
2.5.1. Optimización monoobjetivo	24
2.5.2. Optimización multiobjetivo	25

2.6. Heurística	26
2.7. Metaheurística	26
2.8. Algoritmos Evolutivos	27
2.8.1. Conceptos Generales	27
2.8.2. Algoritmo Genético	30
2.8.3. Algoritmo NSGAII (<i>Non-Dominated Sorting Genetic Algorithm II</i>)	30
2.9. Problemas de optimización en RDA	37
2.9.1. <i>Pipe Optimizing</i>	37
2.9.2. <i>Pumping Schedule</i>	38
2.10. Trabajo relacionado	41
3. Metodología de desarrollo	44
4. Desarrollo	48
4.1. Concepción del proyecto	48
4.2. Funcionalidades	49
4.3. Planificación	49
4.4. Requisitos	51
4.5. Diseño	53
4.5.1. Detalles de implementación	61
4.6. Implementación	65
4.7. Pruebas	72
5. Evaluación de la solución	77
5.1. Metodología de evaluación	77
5.2. Diseño del estudio de caso	77
5.2.1. Elección del caso	78
5.2.2. Objetivos de la investigación	78
5.2.3. Características a evaluar	78
5.2.4. Protocolo para conducir el estudio de caso	79
5.2.5. Unidad de análisis	79
5.3. Consideraciones preliminares	79
5.4. Recolección de datos	80
5.5. Análisis de datos	81

5.5.1. Funcionalidad	82
5.5.2. Usabilidad	83
5.5.3. Utilidad	83
5.5.4. Utilidad del manual de usuario	85
5.6. Conclusiones del estudio	86
6. Conclusiones Y Trabajos Futuros	87
6.1. Conclusiones	87
6.2. Trabajo futuro	88
Bibliografía	90
Anexos	
A: Documento de especificación de requisitos	95
B: Documento de diseño	159
C: Manual de usuario	205
D: Documento de casos de prueba	229
E: Encuesta para la evaluación de la aplicación	279
F: Respuestas de la encuesta aplicada para evaluar la aplicación	289

ÍNDICE DE FIGURAS

	página
2.1. Metodología iterativa e incremental	16
2.2. Componentes físicos de un sistema de distribución de agua	22
2.3. Gráfica objetivo versus generación para un problema	24
2.4. Ejemplo de dominancia y Óptimo de Pareto	25
2.5. Ejemplo frente de Pareto	26
2.6. Cruzamiento en un punto	29
2.7. Mutación aleatoria	29
2.8. Pseudocódigo Algoritmo Evolutivo	30
2.9. Pseudocódigo de la función de remplazo utilizada en el algoritmo NS-GAII	31
2.10. Procedimiento NSGAII	32
2.11. Frentes no dominados	33
2.12. Pseudocódigo de la función de ordenamiento utilizada en NSGAII . .	34
2.13. Cálculo de la densidad de estimación al rededor de la solución i . . .	35
2.14. Pseudocódigo de la función de asignación de densidad	36
2.15. Representación de la solución del problema monoobjetivo <i>Pipe Optimizing</i>	38
2.16. Representación de la solución problema multiobjetivo <i>Pumping Schedule</i>	41
4.1. Arquitectura física monolítica	55
4.2. Arquitectura lógica Modelo-Vista-Controlador	56
4.3. Diagrama de clases de la abstracción de la red resumido	57
4.4. Diagrama de clases modulo metaheurística.	58
4.5. Código del método <i>runSingleStep</i> utilizado con GA y NSGAII	59
4.6. Diagrama de clases para la simulación hidráulica	61
4.7. Interfaz de clase <i>Registrable</i>	62
4.8. Diagrama de actividades de la aplicación	64
4.9. Ventana principal de la aplicación donde se visualiza la red	67
4.10. Ventana de descripción del problema.	67
4.11. Ventana de configuración del problema.	68
4.12. Ventana de configuración de parámetros del operador <i>UniformSelection</i>	68

4.13. Ventana del retroalimentación mostrada durante la ejecución	69
4.14. Ventana de resultados generada cuando termina la ejecución para el problema monoobjetivo Pipe Optimizing	69
4.15. Ventana de simulación hidráulica utilizando los valores del archivo de red	70
4.16. Uso de anotaciones en la clase que hereda <i>Registrable</i> para construir la interfaz de configuración del problema	71
5.1. Gráfico circular de los resultados de la evaluación de funcionalidad de la aplicación	82
5.2. Gráfico circular de los resultados de la evaluación de utilidad de la aplicación	84
5.3. Gráfico circular de los resultados de la evaluación de la utilidad del manual de usuario	85

ÍNDICE DE TABLAS

	página
4.1. Planificación de las iteraciones	50
4.2. Actividades y cambios de la fase de requisitos durante cada iteración	51
4.3. Especificación del requisito de usuario RU004.	52
4.4. Especificación del requisito de usuario RU002.	52
4.5. Especificación del requisito de usuario RU020.	53
4.6. Actividades y cambios en la fase de diseño durante cada iteración . .	54
4.7. Actividades fase de implementación	65
4.8. Especificación caso de prueba manual MT001	73
4.9. Especificación caso de prueba manual MT002	73
4.10. Especificación caso de prueba manual MT003	74
4.11. Especificación caso de prueba manual MT013	74
4.12. Especificación caso de prueba manual MT014	75
4.13. Especificación caso de prueba manual MT015	75
4.14. Especificación caso de prueba manual MT016	76
5.1. Resultados de la evaluación de la funcionalidad	82
5.2. Escala de rangos de <i>SUS Score</i>	83
5.3. Resultados de la evaluación de la utilidad	84
5.4. Resultados de la evaluación de la utilidad del manual de usuario . .	86

RESUMEN

La escasez de agua potable es un problema a nivel mundial. Cada día aumenta la necesidad de utilizar eficientemente los recursos hídricos disponibles. Sin embargo, la optimización de todos los procesos involucrados en su gestión es una tarea compleja debido a que implica el modelamiento de variables físicas que se relacionan de manera no lineal [1]. Junto a lo anterior, se tiene que los encargados de implementar los sistema de distribución de agua (RDA), no cuentan con las suficientes herramientas y tiempo para la correcta gestión de estos sistemas.

Es por ello que este proyecto busca llevar a cabo el desarrollo de una herramienta de escritorio extensible que permita la optimización de los procesos de diseño y operación en RDA. Con el fin de valorar la capacidad del programa, se han implementado dos problemas los cuales son la optimización del diseño de RDA basado en la selección del diámetro de tuberías; y la optimización del régimen de bombeo a través de un enfoque multiobjetivo.

La metodología utilizada para la implementación es la iterativa e incremental. Ésta metodología fue escogida principalmente por la documentación que su aplicación genera, así como la retroalimentación e importancia de la participación del cliente en el desarrollo del proyecto.

La implementación se realizó utilizando el lenguaje de programación Java. Para la implementación de la herramienta fueron importantes 2 funcionalidades de este lenguaje las cuales son *Java Reflection* y *Java annotation*. Éstas nos permitieron lograr un mejor desacoplamiento entre las clases facilitando la extensión de la aplicación para agregar nuevos algoritmos metaheurísticos y problemas.

Para realizar el proceso de evaluación de la solución se utilizaron los estudios de caso, los cuales permiten analizar el objeto de estudio en un contexto real. Este estudio se realizó sobre profesionales con conocimientos en hidráulica y computación, dando como resultado una favorable aceptación de la herramienta.

Al finalizar el proceso de desarrollo y evaluación, se concluye que los objetivos planteados fueron logrados exitosamente. También se presentan una serie de sugerencias que pueden ser implementadas para mejorar la herramienta como la implementación de nuevos algoritmos y problemas; métodos para comparar algoritmos metaheurísticos, entre otros.

1. Introducción

Este capítulo tiene como objetivo presentar el contexto, el problema, la propuesta de solución, los objetivos y el alcance del proyecto. Así como también, entregar antecedentes sobre los trabajos relacionados al tema.

1.1. Contexto del proyecto

La escasez de agua potable es sin duda una problemática a nivel mundial y la optimización de los sistemas que permiten su distribución es cada día más relevante. Existe una serie de problemáticas asociadas a la determinación de las condiciones óptimas de operaciones y las características adecuadas para su construcción.

Las redes de agua potable (RDA) son redes que pueden ser muy extensas y complejas. Forman parte de la estructura principal de cualquier ciudad. Deben ser capaces de adaptarse a los cambios y asegurar niveles mínimos de servicios durante las 24 horas del día [2]. Adicionalmente, dependiendo de su topología, las RDA integran sistemas de bombeo que requieren gran cantidad de energía en horarios determinados.

La optimización de estos sistemas, a la vez, involucra la participación de múltiples criterios que deben ser tomados en cuenta a la hora de decidir. Sin embargo, la incorporación de éstos, involucra la generación de modelos cada vez más complejos [3].

Por lo anteriormente mencionado, esta área, ha llamado la atención de muchos investigadores que han creado diversos métodos para resolver la problemática desde diferentes enfoques. Sin embargo, aún existen muy pocas aplicaciones computacionales que permitan emplear los nuevos modelos y técnicas de forma práctica. Ésto

supone un gran problema para los interesados en aplicar estos conocimientos en un contexto real. Generalmente se trata de personas instruidas en temáticas relacionadas con la hidráulica pero que poseen un escaso manejo de técnicas computacionales de optimización.

En este trabajo se pretende dar respuesta a esa necesidad creciente a través del diseño e implementación de una aplicación de escritorio. Este nuevo sistema, permitirá a los usuarios resolver dos de los principales problemas en la optimización de RDA. En el caso de los problemas con solo un objetivo se utilizará un Algoritmo Genético, mientras que en los problemas multiobjetivos se utilizará NSGAII.

1.2. Presentación del problema

Los encargados de implementar sistemas de distribución de agua potable, no cuentan con suficientes herramientas y tiempo para su correcta gestión. Por lo tanto, no es posible utilizar los recursos asociados de forma eficiente. Además, las herramientas existentes no satisfacen las necesidades de usabilidad y costo, debido a que son poco intuitivas y de pago.

El escoger las especificaciones de una red de agua potable es una tarea difícil debido a que hay que evaluar el rendimiento general del sistema en función de un conjunto de variables que se mueven en un rango muy elevado de posibilidades. Debido a esto, el uso de herramientas que optimicen la selección de estas características puede ayudar considerablemente a reducir costos operaciones y de inversión.

Finalmente, es importante destacar que la construcción de un sistema que permita realizar la optimización de RDA es compleja. Necesita del conocimiento técnico de expertos en el área de hidráulica y computación. Involucra el trabajo con programas de simulación computacional que modelan las características de los sistema de agua bajo presión y de algoritmos metaheurísticos que los subordinen.

1.3. Objetivos

Para abarcar la problemática se fijan los siguientes objetivos.

Objetivo general

- Diseñar y desarrollar una aplicación extensible de escritorio para optimizar el diseño y operación de una red de distribución de agua.

Objetivos específicos

1. Diseñar software orientado a la optimización de RDA basado en la arquitectura lógica del framework multiobjetivo jMetal [4].
2. Implementar un algoritmo metaheurístico de optimización monoobjetivo para aplicar al problema de diseño de RDA.
3. Implementar un algoritmo metaheurístico de optimización multiobjetivo para aplicar al problema de Régimen de bombeo en RDA.
4. Diseñar e implementar la interfaz gráfica del sistema de optimización de redes de agua potable desarrollado durante este proyecto.

1.4. Propuesta de solución

La solución que se propone para abordar el problema consiste en el desarrollo de una aplicación de escritorio extensible que permita optimizar procesos de diseño y operación en RDA. Este software entrega a sus usuarios la posibilidad de agregar nuevos problemas y algoritmos si se consideran necesarios.

Los problemas que se abordan en el contexto de optimización de redes de agua potable serán:

- **Problema de diseño:** Este tipo de problema busca optimizar las configuraciones y la disposición de los elementos que conforman la red previa a su construcción [5].
- **Problema de operación:** Los problemas de operación buscan optimizar las configuraciones de una red ya construida [5].

Con el fin de valorar la capacidad del programa, dos problemas hidráulicos son implementados. Cada uno con un algoritmo afín según sus características.

La selección de estos problemas se llevo a cabo teniendo en cuenta la necesidad de los interesados en el estudio de redes de distribución de agua. Específicamente, los participantes del proyecto de investigación *Optimization of real-world water distribution systems and hydraulic elements using computational fluid dynamics (cfd) and evolutionary algorithms* financiado por la Agencia Nacional de Investigación y Desarrollo ANID, Chile y sus colaboradores en la Universidad Politécnica de Valencia, España.

Según lo anteriormente expuesto, los problemas seleccionados son los siguientes:

1. Problema de diseño de RDA basado en el costo de tuberías. El cual consiste en la optimización de los costo de inversión variando el diámetro de las tuberías (*Pipe Optimizing*) [6, 2]. Este problema sera abordado utilizando el Algoritmo Genético (GA).
2. Problema de operación basado en el régimen de bombeo. Éste consiste en la optimización de costos energéticos y el costo de mantenimiento de las bombas (*Pumping Schedule*) [3]. Este problema se aborda desde el enfoque multiobjetivo utilizando el algoritmo *Non-Dominated Sorting Genetic Algorithm*, versión II (NSGAII).

Adicionalmente, se genera una guía para que los usuarios puedan implementar nuevos problemas, algoritmos y operadores.

1.5. Alcances

Los alcances propuestos para este proyecto son los siguientes:

- Esta herramienta sólo puede ser ejecutada en equipos con el sistema operativo Window de 64bits. Esta limitación se debe a que se realizan llamadas a librerías nativas que fueron compiladas para sistemas de 64bits.
- El sistema permite la carga y la visualización de la red gráficamente.
- El sistema permite visualizar la configuración básica almacenada en el archivo de configuración de red.
- Este proyecto no contempla la creación de la red desde el programa a desarrollar.

- El archivo de configuración de red debe ser creado utilizando la aplicación Epanet y guardado con la extensión inp.
- El sistema implementa por defecto dos algoritmos. Uno de ellos monoobjetivo y el otro multiobjetivo. El algoritmo monoobjetivo corresponde al Algoritmo Genético (GA) mientras que el multiobjetivo a *Non-Dominated Sorting Genetic Algorithm II* (NSGAI).
- El sistema proporciona dos problemas ya implementados uno monoobjetivo y el otro multiobjetivo. El Algoritmo Genético se utiliza para generar soluciones para el problema monoobjetivo con el fin de optimizar el costo de inversión de las tuberías. Por otro lado, el algoritmo NSGAI aborda el problema multiobjetivo con el objetivo de optimizar los costos energéticos y costo de mantenimiento de los equipos de bombeo.
- El sistema permite visualizar él o los resultados obtenidos al finalizar la ejecución del algoritmo.
- El sistema permite utilizar una solución obtenida, a través de los algoritmos metaheurísticos, para generar un nuevo archivos de configuración de red.
- El sistema permite generar archivos con las soluciones obtenidas para cada problema, es decir, el valor de los objetivos y las variables de decisión involucradas.
- El sistema permite graficar visualmente las soluciones en un plano cartesiano.
- La gráfica únicamente está disponible en problemas con uno o dos objetivos.

2. Marco Teórico

En este capítulo se presentan los conceptos que serán necesarios para la realización del proyecto.

2.1. Metodología de desarrollo

La metodología de desarrollo es un conjunto de actividades relacionadas entre sí y que se realizan en cierto orden definiendo un flujo de trabajo que permite llevar a cabo el desarrollo de un proyecto [7].

Una de estas metodologías es la iterativa e incremental. Ésta metodología, lleva a cabo el desarrollo de un proyecto de software dividiéndolo en iteraciones que generan un incremento. Este incremento contribuye en el desarrollo del producto final [8].

Cada iteración se compone de las fases de análisis, diseño, implementación y pruebas como se muestra en la Figura 2.1. La fase de análisis se encarga de llevar a cabo la obtención y definición de los requerimientos del software. Durante la etapa de diseño se realiza la conceptualización del software basado en los requerimientos definidos anteriormente. Durante la implementación se codifican las funcionalidades siguiendo las directivas establecidas durante el diseño, con el fin de satisfacer los requerimientos. Y finalmente, durante la fase de pruebas, se valida y verifica la correctitud de las funcionalidades implementadas, así como el cumplimiento de los requisitos.

El hecho de llevar a cabo un desarrollo iterativo permite la obtención de retroalimentación del producto que se está desarrollando y de esta manera poder refinar el trabajo en etapas posteriores del desarrollo [9, 10, 11, 8].

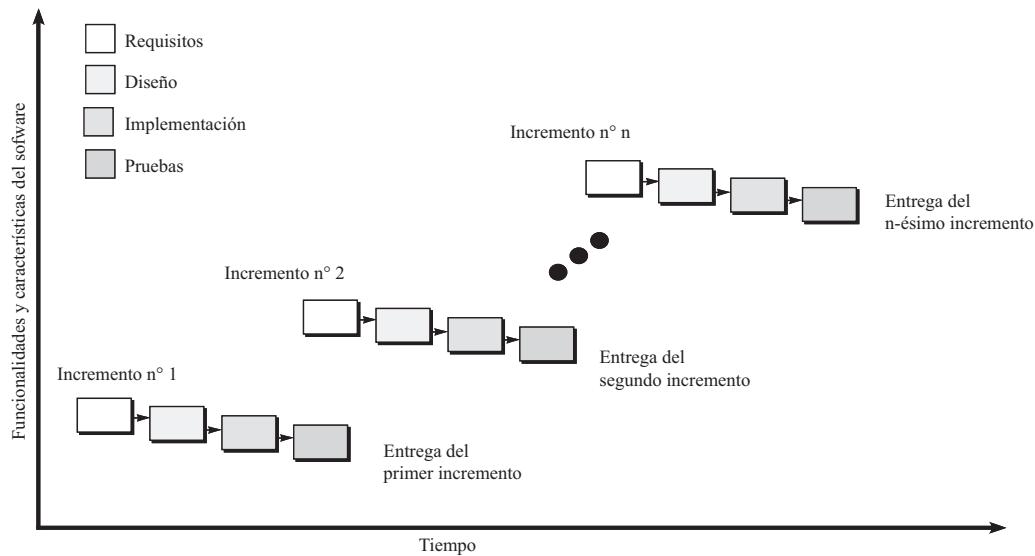


Figura 2.1: Metodología iterativa e incremental [7]

2.2. Metodología de evaluación

La metodología de evaluación define el conjunto de actividades a realizar con el fin de analizar o evaluar un proyecto, un grupo de personas, un producto, etc.

En ingeniería de software los estudio de caso [12] pueden ser usados para analizar un sistema en un contexto real con el objetivo de responder la pregunta de investigación planteada. Esta metodología de evaluación considera aspectos formales para obtener evidencia. Las actividades para llevar a cabo el estudio de caso se pueden dividir en las siguientes etapas:

1. Diseño del estudio de caso:

- (I) **Elección del caso:** Consiste en definir el objeto a estudiar. Éste puede ser un proceso, producto, grupo de persona, etc.
- (II) **Definición de objetivos experimentales:** Consiste en indicar cual es el objetivo de la investigación a realizar, si es describir, evaluar o explicar algún suceso.
- (III) **Definición de un protocolo para conducir el estudio de caso:** Consiste en escoger las pautas para llevar a cabo el estudio de caso, que instrumentos serán utilizados para recolectar datos y cómo se realiza el análisis de éstos.

- (iv) **Definición de características a evaluar:** Consiste en establecer una lista de características a evaluar del elemento sobre el que se aplica el estudio de caso.
 - (v) **Definición de sujetos de prueba:** Consiste en indicar cuál es la fuente de datos a ser utilizada para el estudio de caso, estas pueden ser personas, datos ya recolectados, etc.
2. **Recolección de datos:** Consiste en aplicar el estudio de caso en un conjunto de sesiones no controladas sobre los sujetos de pruebas definidos.
3. **Análisis de los datos recolectados:**
- (I) **Aplicación de herramientas de obtención de evidencia empírica:** Consiste en la utilización de métodos o técnicas con el fin de obtener evidencia empírica a partir de los datos recolectados.
 - (II) **Análisis y evaluación de datos empíricos:** Consiste en analizar la evidencia y evaluar la validez de los resultados obtenidos.
4. **Reportar los resultados del estudio de caso:** Presentar los resultados y las conclusiones del estudio de caso.

2.3. Herramientas para el desarrollo del software

A continuación se presentan las herramientas utilizadas para el desarrollo del proyecto.

2.3.1. Lenguaje de programación Java

Java es un lenguaje de programación de alto nivel orientado a objetos y de propósito general. Fue anunciado por Sun Microsystems en Mayo de 1995 [13]. Un programa Java se ejecuta sobre la maquina virtual llamada la *Java Virtual Machine*, la cual le da a este lenguaje la característica de ser multiplataforma. Adicionalmente, Java incorpora el soporte para multi-hilos, una poderosa herramienta que permite la ejecución de distintas instrucciones de código al mismo tiempo [14]. Ademas. este lenguaje también posee una característica conocida como el recolector de basura, que se encarga de limpiar la memoria de objetos que ya no están siendo utilizados.

La elección de Java como lenguaje de programación para desarrollar este proyecto fue un requerimiento de los investigadores que participan del proyecto ya que permite la incorporación directa de algunas herramientas que han sido diseñadas.

A continuación se describen las funcionalidades de Java que han sido más importantes para desarrollar este trabajo.

Java Reflection

Característica de Java que permite que un programa se auto examine. Esta característica está disponible a través de la *Java Reflection API*, la cual cuenta con métodos para obtener los metadatos de las clases, métodos, constructores, campos o parámetros. Esta API también permite crear nuevos objetos cuyo tipo era desconocido al momento de compilar el programa [15].

Java Annotation

Característica de Java para agregar metadatos a elementos del lenguaje como las clases, métodos, parámetros, etc [16]. Las anotaciones no tienen efecto directo sobre el código. Sin embargo, combinadas con *Java Reflection* permiten realizar una serie de tareas muy útiles como crear nuevos objetos cuyo tipo no conocemos en tiempo de compilación.

Las anotaciones están presentes en varios lenguajes como Python y C#, siendo en este último llamadas Atributos. Generalmente, los lenguajes que incorporan anotaciones también implementan la técnica llamada reflexión.

A continuación se presenta un ejemplo de la anotación `@Override` en Java:

```
@Override  
public void toString(){}

---


```

Polimorfismo

El polimorfismo es la técnica que permite modificar el comportamiento de un elemento del lenguaje [17], sea este una clase, función, método u operador, dependiendo del tipo de dato real con el que se está trabajando. Existen muchas formas de polimorfismo. Las principales son nombrados a continuación usando su nombre en inglés:

- *Universal Polymorphism*: Son aquellas técnicas que trabajar sobre un cualquier tipo de instancia con una estructura común, es por ello que pueden emplear la misma operación sin cambios en el código.
 - *Parametric polymorphism*: Se refiere al uso de una función u objeto sin importar el tipo de instancia sobre la que trabaja, es decir, opera de manera uniforme los distintos tipos que desea operar. Las funciones que cuentan con este tipo de polimorfismo se conoce también como funciones genéricas. A modo de ejemplo se encuentran los genéricos de Java y los *templates* de C++.
 - *Inclusion*: Hace referencia a los subtipos. En donde una clase puede ser usado y manipulado donde quiera que se espere el supertipo de ésta.
- *Ad-hoc polymorphism*: Estas técnicas trabajan sobre un conjunto limitado de tipos que no están relacionados, necesitando manejar cada tipo de manera diferente.
 - *Overloading*: Hace referencia a la reutilización de símbolos o nombres de funciones en más de un contexto. Por ejemplo, algunos lenguajes permiten usar el mismo nombre de función variando el número de argumentos recibidos o el tipo de ellos. También se cuenta la sobrecarga de los operadores que permiten algunos lenguajes como C++.
 - *Coercion*: Este tipo de polimorfismo hace referencia a la conversión automática de tipos. Por ejemplo, al sumar un valor entero y un real en algunos lenguaje de programación, el valor entero es convertido a real antes de aplicar el operador de la suma.

Java incorpora variaciones de los tipos anteriormente mencionados de polimorfismo.

2.3.2. Patrones de diseño

En esta sección se describen los patrones de diseño que fueron más importantes durante la realización del proyecto.

Inversión de control

La Inversión de Control [18] es un principio de ingeniería de software en que el control se transfiere desde el programador al programa, permitiendo así sistemas con menor acoplamiento. Utilizando este principio se establece un flujo de ejecución en la aplicación en el cual se incorporan las llamadas al código implementado por el cliente. Esta técnica es ampliamente usada en los *frameworks*. Dos ejemplos de framework que usan esta técnica son Spring en Java y Angular en JavaScript [19].

Inyección de dependencias

La Inyección de Dependencias (DI) [18], es un patrón de diseño que permite reducir el acoplamiento entre los componentes dejando la tarea de crear las dependencias de un objeto en manos de un externo, el cual podría ser un modulo dentro del programa, para posteriormente enviarla a la instancia que la requiera. Este es una de las técnicas utilizadas para implementar el principio de inversión de control en un sistema. De acuerdo a [19] la inyección de dependencias presenta los siguientes problemas [19]:

- Las dependencias inyectadas deben ser compatibles con la abstracción definida en la clase, es decir, la instancia a inyectar debe ser un subtipo de la clase esperada.
- Las clases que se inyectan deben implementar el comportamiento que espera la clase que hace uso de esta dependencia.

2.3.3. Epanet Programming Toolkit

Librería de enlace dinámico que permite realizar simulaciones computaciones del comportamiento del agua en RDA. Esta librería es parte de Epanet, un software que permite simular el comportamiento hidráulico y la calidad del agua en redes de distribución de aguas compuesta por tuberías, nodos, bombas, válvulas y tanques de almacenamiento [20]. Fue creada por la agencia EPA de EE.UU. La librería cuenta con un conjunto de funciones para realizar simulaciones desde diferentes entornos de desarrollo como C, C++, VB, Java, etc [21].

2.4. Hidráulica urbana

Este trabajo se centra en la implementación de una aplicación en el área de hidráulica. Es por ello, que es necesario comprender algunos conceptos básicos que se describen a continuación:

2.4.1. Conceptos de básicos de hidráulica

A pesar de que existe una gran diversidad de términos hidráulicos, sólo se describen los estrictamente necesarios para entender el funcionamiento del programa.

- **Presión:** Fuerza ejercida sobre una superficie. En este caso se refiere a la fuerza que el agua ejerce durante su recorrido. Generalmente, se expresa como una unidad del sistema técnico a través de metros columna de agua (mca) [22].
- **Caudal:** Cantidad de agua que se mueve a través de un segmento de la red. Se expresa en el sistema internacional como metros cúbicos por segundo (m^3/s) [22].
- **Curva de consumo:** Patrón de consumo de agua en un periodo de tiempo. Generalmente el agua es demandada de forma irregular durante el tiempo debido principalmente a las diferentes actividades que la población realiza durante el día. Así, por ejemplo, el consumo de agua aumenta en los horarios de la mañana y tarde [22].

2.4.2. Red de distribución de agua

Conjunto de elementos enlazados de tal manera que permite suministrar cierta cantidad de agua a una presión establecida [22, 23]. En la Figura 2.2 se puede apreciar los componentes que conforman la red de agua potable.

Componentes de la red

Nodos de consumo: Son puntos de extracción de agua. Los niveles de consumo son afectados por la curva de demanda. Cada uno de estos puntos puede requerir diferentes presiones dependiendo de la actividad que genere dicha demanda. En una ciudad, la mayor parte de estos nodos corresponden a viviendas.

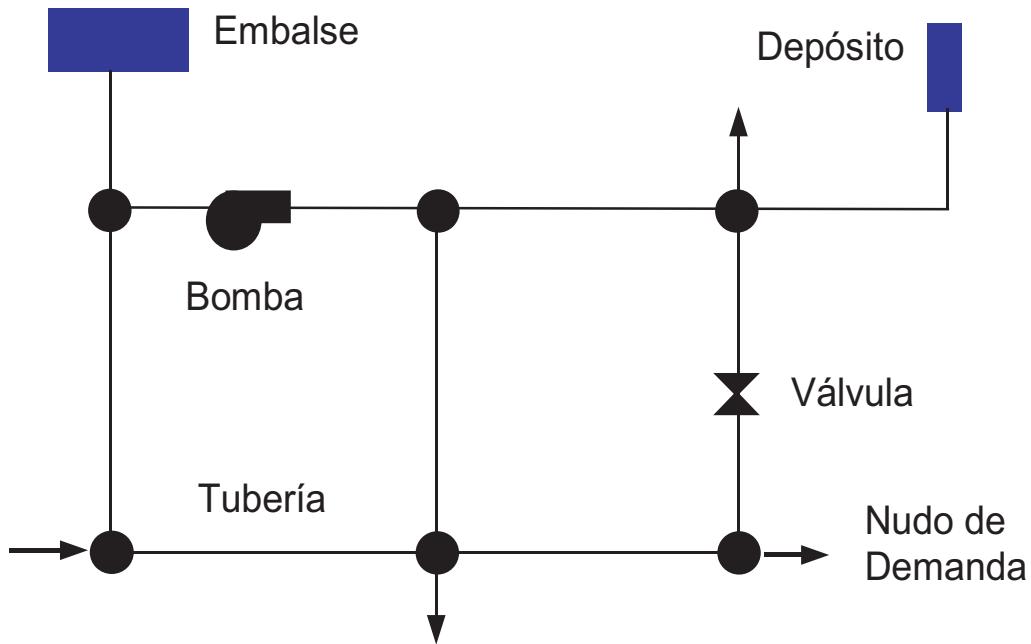


Figura 2.2: Componentes físicos de un sistema de distribución de agua [20]

Reservorio: Fuente de alimentación externa desde la que se extrae el agua para alimentar la red, esta fuente puede ser un embalse, un río, un lago, etc.

Bombas: Componentes que permiten aportar energía al sistema con el fin de impulsar el agua a través de la red. Se encuentran agrupados en estaciones de bombeo. Y su consumo energético representa uno de los costos más significativos durante las operaciones en RDA. Por lo tanto, su estudio es verdaderamente importante.

Depósito: Son elementos con la capacidad de almacenar agua. Generalmente se utilizan para disminuir el trabajo que realizan los equipos de bombeo y permiten aumentar la disponibilidad del recurso en casos de emergencia.

Tuberías: Es el medio por el cual transita el agua desde un lugar a otro. Existen diferentes tipos de tuberías según su material, diámetros y largos. La selección de éstos constituye un problema importante en el diseño de RDA.

2.5. Optimización

Optimización consiste en maximizar o minimizar un conjunto de funciones, modificando una serie de variables, conocidas como las variables de decisión o independientes. Ésta puede ser expresada matemáticamente de la siguiente forma:

$$f_1(x), f_2(x), \dots, f_N(x), \quad x = (x_1, \dots, x_d) | x \in X$$

sujeto a una serie de restricciones

$$h_j(x) = 0, \quad j = 1, 2, \dots, J$$

$$g_k(x) \leq 0, \quad k = 1, 2, \dots, K$$

siendo f_1, \dots, f_N funciones objetivos; x_1, \dots, x_d variables de decisión, pertenecientes al espacio de búsqueda X ; y h_j junto con g_k , una serie de restricciones [24]. De acuerdo a la cantidad de funciones objetivos (N) que se tenga, se establece que si $N = 1$ la optimización es **monoobjetivo**, mientras que para $N \geq 2$ se conoce como **multiobjetivo** [24]. Cada uno de estos tipos de problemas debe ser abordado con metodologías específicas para su propósito ya que sus objetivos son diferentes; por un lado, los problemas monoobjetivos persiguen encontrar una única solución, mientras que los problemas multiobjetivos se enfocan en determinar un conjunto de soluciones llamado Frontera de Pareto que será descrito en el apartado 2.5.2. En este punto se debe tener en cuenta que los objetivos planteados deben encontrarse en contradicción. Es decir, para que un objetivo mejore debe existir un empeoramiento en alguno de los otros.

Debido a la definición de las restricciones es posible dividir el espacio de búsqueda en dos regiones [25]:

- Soluciones factibles: Compuesto por los elementos pertenecientes al espacio de búsqueda que satisfacen todas las restricciones.
- Soluciones no factibles: Integrado por aquellos elementos que no cumplen todas las restricciones.

2.5.1. Optimización monoobjetivo

La optimización monoobjetivo es aquella que busca maximizar o minimizar una única función. Como resultado del proceso de optimización se espera encontrar una sola solución la cual debiese ser lo mas cercano posible al valor óptimo de la función. Para resolver problemas que utilizan este enfoque existen diversos algoritmos. Los algoritmos más conocidos para este tipo de optimización se clasifican bajo las siguientes categorías [26]:

- Basados en una sola solución: Estos son métodos que buscan mejorar una única solución. Entre los algoritmos de esta categoría se encuentran *Simulated Annealing*, *Tabu Search*, *Greedy Randomized Adaptive Search Procedure* (GRASP), *Local Search*, etc.
- Basados en población: Métodos en los cuales se realiza la búsqueda de la solución óptima a partir de un conjunto de soluciones [27]. Las principales categorías de este tipo de métodos son aquellos relacionados a los procesos evolutivos, clasificados como algoritmos evolutivos y los relacionados en el comportamiento colectivo de las entidades, conocidos como inteligencia de enjambre.

La Figura 2.3 muestra un ejemplo de la mejora de las soluciones para el Algoritmo Genético.

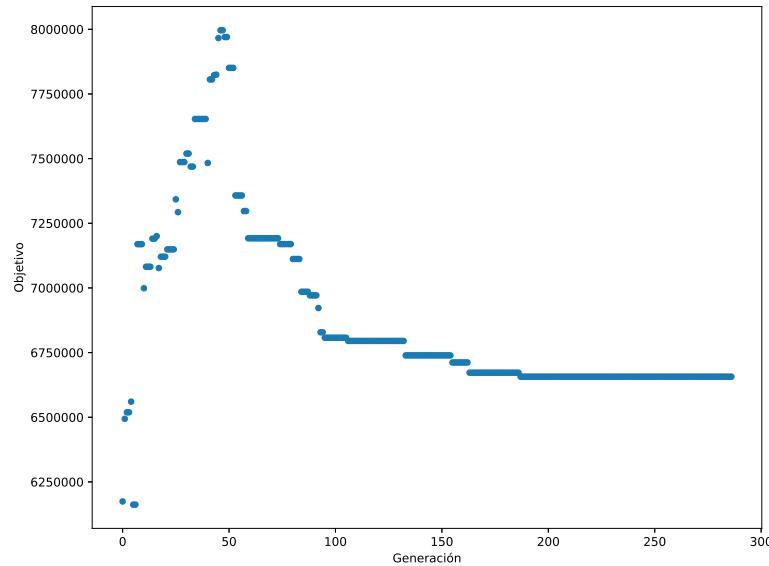


Figura 2.3: Gráfica objetivo versus generación para un problema

2.5.2. Optimización multiobjetivo

Como resultado de un proceso de optimización multiobjetivo no existe una única solución a un problema, sino que se tiene un conjunto de soluciones. Es por ello que a continuación se presentarán una serie de criterios que permitirán analizar y determinar el conjunto de soluciones optimas. Los criterios son los siguientes [28]:

Dominancia de Pareto

Sean u y v vectores pertenecientes a \Re^n , se dice que u domina a v (se denota como $u \preceq v$) si, y sólo si (en el caso de minimización):

$$\forall i \in \{1, 2, \dots, n\} | f_i(u) \leq f_i(v) \wedge \exists j \in \{1, 2, \dots, n\} | f_j(u) < f_j(v)$$

Es decir, para que una solución domine a otra, cada uno de sus objetivos debe ser mejores o iguales y al menos en uno de ellos este debe ser mejor.

El ejemplo de la Figura 2.4 muestra los vectores A,B,C,D de los cuales C y B dominan a D, y A domina a C y D. Nótese que C y B no son dominantes.

Óptimo de Pareto

Una solución u es un óptimo de Pareto si no hay otra solución v en el espacio Ω , tal que v domine a u , es decir, para $u, v \in \Re^n \nexists v \preceq u$. Los óptimos de pareto también se conocen bajo el nombre de solución no-dominada. En la Figura 2.4 el vector A sería un óptimo de Pareto.

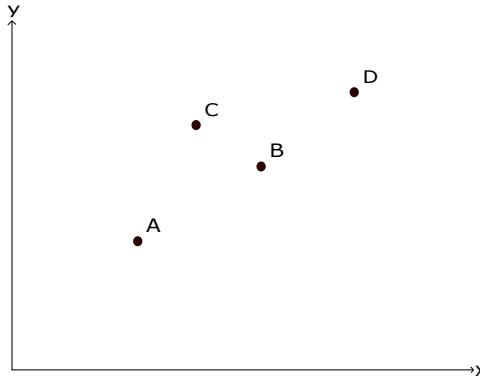


Figura 2.4: Ejemplo de dominancia y Óptimo de Pareto

Frontera de Pareto

La frontera de Pareto es el conjunto de todas las soluciones no dominadas las cuales componen las soluciones óptimas al problema multiobjetivo. En la Figura 2.5 los puntos rojos componen la frontera de Pareto.

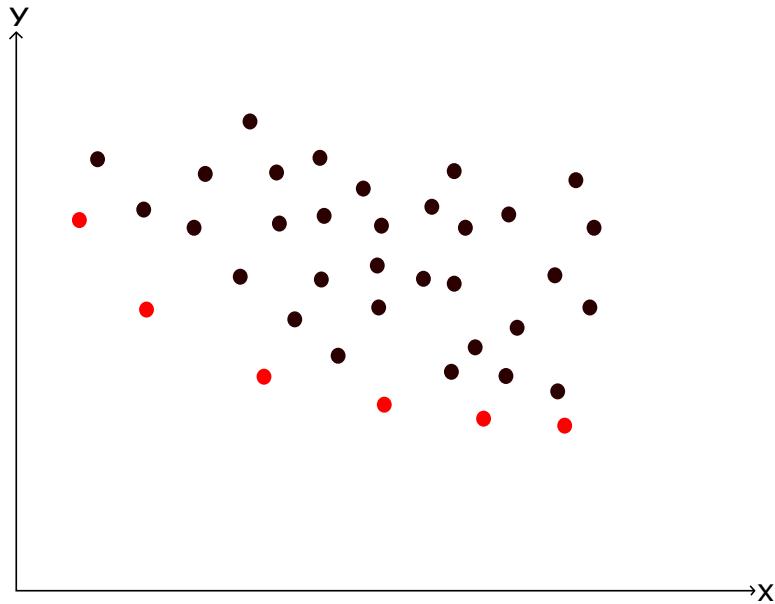


Figura 2.5: Ejemplo frente de Pareto

2.6. Heurística

Es el conocimiento que se tiene del problema el cual permite acotar la búsqueda de las soluciones en espacios de búsqueda de gran tamaño que hacen inviable la aplicación de técnicas deterministas por el costo de tiempo que implican. Con la utilización de estas técnicas se espera encontrar soluciones buenas en un tiempo razonable, pero esto no esta garantizado [24, 29].

2.7. Metaheurística

Algoritmos que permiten resolver un amplio rango de problemas de optimización empleando técnicas con algún grado de aleatoriedad para encontrar soluciones a un problema. Estos algoritmos no garantizan que la solución encontrada sea la

óptima, pero permiten obtener generalmente aproximaciones a ésta. La diferencia entre heurísticas y metaheurísticas, es que esta última puede ser aplicado a un amplio conjunto de problemas sin necesidad de realizar grandes cambios en el algoritmo, mientras que las heurísticas generalmente son aplicadas a un dominio específico [24, 26, 30].

2.8. Algoritmos Evolutivos

Conjunto de algoritmos inspirado en la teoría de la evolución de Darwin acerca de la capacidad de la naturaleza para evolucionar seres vivos bien adaptados a su entorno. Estos algoritmos hacen uso de diversos mecanismos entre los que se encuentra la selección, mutación y cruzamiento sobre los individuos de una población con el fin de generar una nueva generación de individuos [26].

2.8.1. Conceptos Generales

Individuo

Representa una solución al problema. Los individuos pueden ser representados de diversas manera. El proceso mediante se realiza este mapeo, se denomina codificación. Se ha demostrado que la selección de una codificación u otra influye directamente en los resultados dependiendo del problema estudiado, entre éstas se encuentra la representación binaria (1 y 0), la real (utilizando números reales), etc. Para la representación binaria cada variable se codifica como un conjunto de bits, lo cual forma una cadena binaria. Por ejemplo, la representación binaria de la solución (2, 4, 6, 8) formada por enteros de 4 bits correspondería a

0010 0100 0110 1000

En cambio, para la representación real esta se presenta como un vector, en donde cada valor que forma este vector pertenece a los números reales, es decir,

$$v = (x_1, x_2, \dots, x_n), \text{ en donde } v \in \mathbb{R}^n$$

Población

Conjunto de individuos (soluciones) candidatos sobre los cuales opera el algoritmo. Durante cada iteración del algoritmo se generan nuevas soluciones que son agregadas a la población a la vez que se remueven otras según cada estrategia [31].

Etapas de un algoritmo evolutivo

- **Selección:** La selección es un mecanismo utilizado por los algoritmos evolutivos para escoger a los individuos más aptos los cuales serán usados para la reproducción [31]. Existen numerosos algoritmos de selección que pueden ser utilizados en los algoritmos evolutivos.
- **Reproducción:** La reproducción es el proceso en el cual se combinan individuos con el fin de mantener la diversidad de las soluciones y generar las nuevas generaciones.

Operadores

Estos procedimientos permiten generar diversidad en las soluciones y a la vez explotar las mejores características de los individuos.

Operadores generales:

- **Cruzamiento:** El cruzamiento es un mecanismo usado para generar nuevas soluciones a partir de dos o más individuos seleccionados [32, 26]. En la Figura 2.6 se aprecia un cruce entre dos individuos en torno a un punto de cruzamiento.

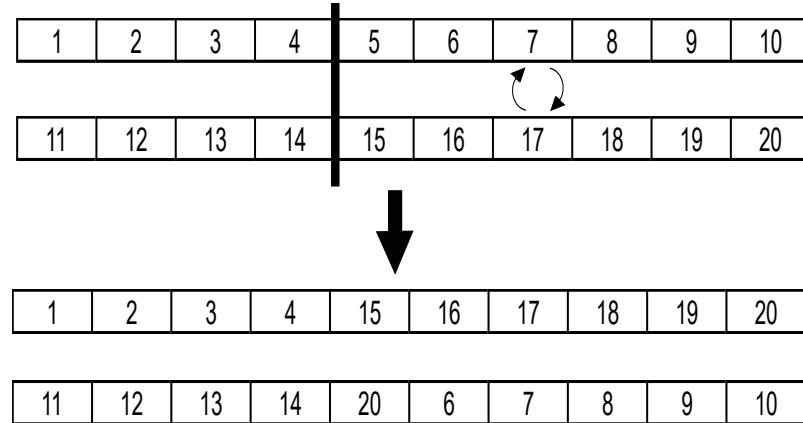


Figura 2.6: Cruzamiento en un punto

- **Mutación:** La mutación es un operador el cual permite mantener la diversidad en la descendencia [26] realizando modificaciones en ciertas partes de la solución. En la FIgura 2.7 se puede ver una mutación aleatoria en un individuo codificados con números enteros.

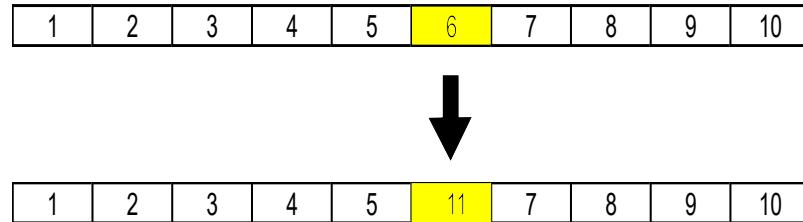


Figura 2.7: Mutación aleatoria

En la Figura 2.8 muestra el pseudocódigo de los pasos generales de un algoritmo evolutivo.

Primero, se crea la población inicial. Después se evalúan los objetivos de dicha población y se itera hasta que la condición de termino haya sido alcanzada. La condición de término puede ser, por ejemplo un máximo numero de evaluaciones o evaluaciones sin mejoras en los resultados. Dentro del ciclo se realiza la selección sobre la población con el fin de determinar las soluciones que serán usadas en los operadores de cruzamiento y mutación. Finalmente, se remplaza la población inicial con la descendiente.

Dentro de los algoritmos evolutivos más utilizados se encuentran el Algoritmo

Algorithm 1: Algoritmo Evolutivo

```

1 población ← crearPoblaciónInicial()
2 evaluarPoblación(población)
3 while la condición de termino no ha sido alcanzada do
4   | poblaciónSeleccionada ← selección(población)
5   | poblaciónDecendiente ← cruzamiento(poblacionSeleccionada)
6   | poblaciónDecendiente ← mutación (poblaciónDecendiente)
7   | poblaciónDecendiente ← evaluarPoblación (poblaciónDecendiente)
8   | población ← remplazar (población, poblaciónDecendiente)
9 end

```

Figura 2.8: Pseudocódigo Algoritmo Evolutivo

Genético (GA), Programación Genética (GP), Evolución Diferencial (DE), Estrategia Evolutiva (ES) y Programación Evolutiva (EP) [33].

Específicamente el GA ha presentado buenos resultados en el área de optimización de RDA [23]. Es por ello que ha sido seleccionado como uno de los algoritmos para ser implementados en este programa.

2.8.2. Algoritmo Genético

El Algoritmo Genético es uno de los algoritmos evolutivos más conocidos y que ha demostrado ser exitoso en una variedad de problemas diferentes. Estos algoritmo hacen uso de los mecanismos de selección y reproducción para generar nuevas soluciones a partir de la combinación de otras [31].

Los individuos en el contexto del Algoritmo Genético son llamados cromosomas, los cuales como se mencionó en la sección de los algoritmos evolutivos, pueden ser representados de diversas maneras.

2.8.3. Algoritmo NSGAII (*Non-Dominated Sorting Genetic Algorithm II*)

El algoritmo NSGAII [34] pertenece a la categoría de algoritmo evolutivo multiobjetivo (MOEA). Este algoritmo al igual que el Algoritmo Genético hace uso de los operadores de selección, cruzamiento y mutación para encontrar un conjunto de soluciones optimas a problemas que cuentan con más de un objetivo. Adicionalmente, NSGAII añade conceptos y operadores adicionales los cuales permiten mejorar su

rendimiento y la calidad de las soluciones obtenidas. NSGAII puede ser implementado siguiendo los mismos pasos de el algoritmo evolutivo mostrados en la Figura 2.8, utilizando la función de remplazo mostrada en la Figura 2.9.

Algorithm 2: Función de remplazo para el algoritmo NSGAII

```

1 Function reemplazar(población, poblaciónDescendiente)
2   unionPoblacion  $\leftarrow$  población  $\cup$  poblaciónDescendiente
3   /*  $F = (F_1, F_2, \dots)$  */ 
4   F  $\leftarrow$  ordenarPorFrentesNoDominados (unionPoblacion)
5   nuevaPoblacion  $\leftarrow$   $\emptyset$ 
6   i = 1
7   /* Hasta que nuevaPoblacion este lleno */ 
8   while ( $|nuevaPoblacion| + |F_i| \leq N$ ) do
9     /* Calcular y asignar la densidad a cada solución del
    frente  $F_i$  */ 
10    asignarDensidad ( $F_i$ )
11    /* Añadir a nuevaPoblacion las soluciones del frente  $F_i$  */ 
12    nuevaPoblacion  $\leftarrow$  nuevaPoblacion  $\cup$   $F_i$ 
13    i = i + 1
14  end
15  /* Ordenar el frente  $F_i$  usando el comparador de densidad */ 
16  ordernar ( $F_i, \prec_n$ )
17  /* Elegir los primeros  $N - |nuevaPoblacion|$  */ 
18  nuevaPoblacion  $\leftarrow$  nuevaPoblacion  $\cup$   $F_i[1 : N - |nuevaPoblacion|]$ 
19  retornar nuevaPoblacion
20 fin
```

Figura 2.9: Pseudocódigo de la función de remplazo utilizada en el algoritmo NSGAII [34]

En la Figura 2.9 se puede ver que el proceso de remplazo dentro del Algoritmo Genético consiste en unir la población actual y la población descendiente (formada por los elementos resultantes del operador de selección y sobre la que se han aplicado el cruzamiento y la mutación) en un solo elemento llamada *unionPoblación*. Luego, esta es enviada a un procedimiento el cual ordena y categoriza la población en diversos frentes de acuerdo al concepto de dominancia de Pareto. Después, de que

la población a sido categorizada se procede a iterar sobre los frentes y añadir sus elementos a una nueva población con el cuidado de no sobrepasar el tamaño de la población deseada (N). En caso de que uno de los frentes no pueda ser añadido en su totalidad por sobrepasar dicho tamaño, se llevará a cabo un proceso por el cual se ordenaran las soluciones en dicho frente basadas en un criterio conocido como densidad de las soluciones. Una vez realizado el ordenamiento, se agregarán las mejores soluciones a la nueva población hasta alcanzar el tamaño deseado N . Se puede ver un ejemplo de este procedimiento gráficamente en la Figura 2.10.

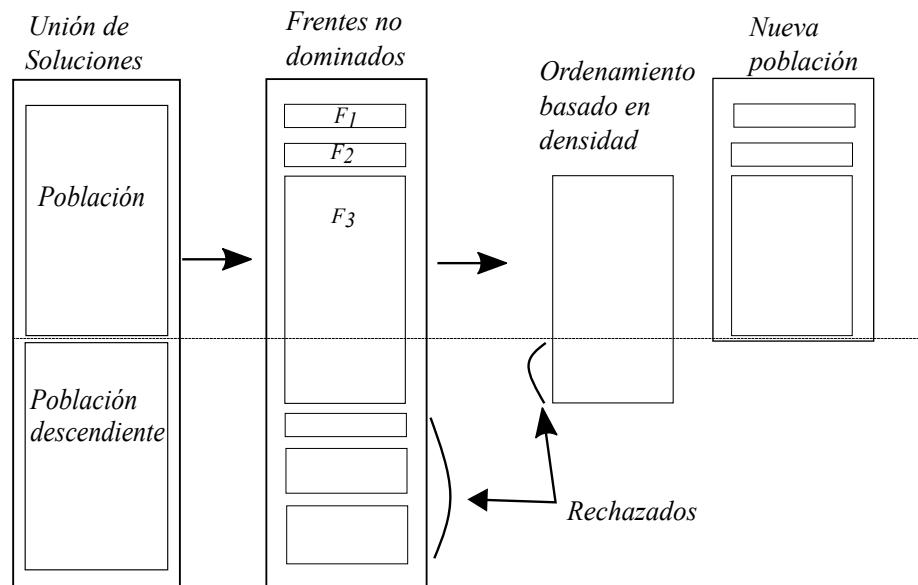


Figura 2.10: Procedimiento NSGAII [34]

A continuación se procederá a explicar los operadores adicionales presentados en [34] y que son utilizados por la función de remplazo del algoritmo NSGAII en la Figura 2.9.

Ordenamiento de soluciones en frentes no dominados

Uno de los procedimientos presentados en la Figura 2.9 consiste en ordenar las soluciones en frentes no dominados. Los frentes no dominados son conjuntos en los que se almacenan las soluciones que no se dominan entre si. En la Figura 2.11 se muestra un ejemplo con tres frentes.

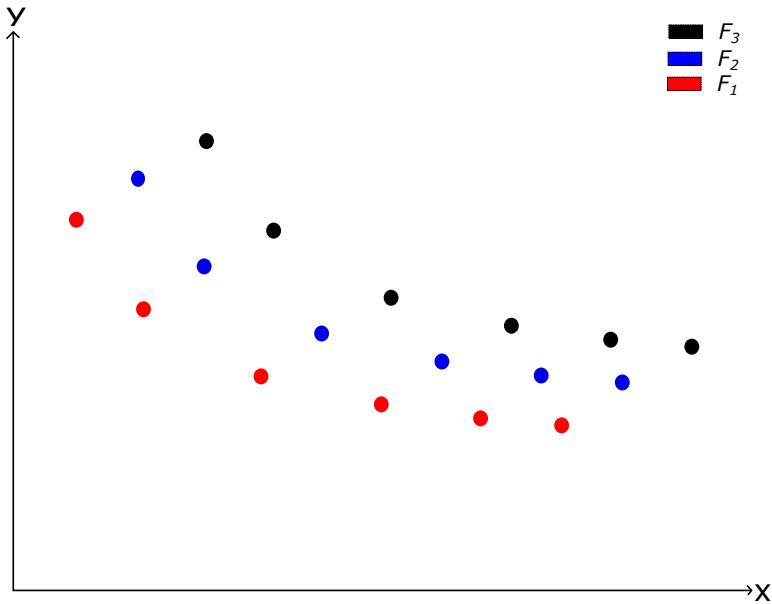


Figura 2.11: Frentes no dominados

El algoritmo para llevar a cabo esto se presenta en la Figura 2.12 y consiste en lo siguiente:

Primero, se debe comparar todas las soluciones entre si utilizando el concepto de dominancia de Pareto. Para ello, cada solución p cuenta con un atributo S_p en el que guarda todas las soluciones a las que domina y un contador n_p que almacena el numero de soluciones que lo dominan a él. Cada vez que se termina de comparar una solución con todas las otras, si su contador n_p es igual a 0, se le asigna a la solución el rango que indica el frente al que pertenece, y se guarda esta en un conjunto F_1 que contiene a todas las soluciones de dicho frente.

Una vez que se han identificado todas las soluciones no dominadas del primer frente se procede a generar el siguiente, para lo cual, por cada solución q almacenada en el conjunto $S_p \in p$ del frente ya conocido, se disminuye en uno su contador n_q . Si el contador n_q de la solución q llega a 0, entonces se le asigna a dicha solución el rango correspondiente y se guarda esta en un conjunto temporal Q con el resto de las soluciones en dicho frente. Cuando se tengan identificadas todas las soluciones, estas se asignan al frente correspondiente F_i .

Finalmente, se repite el procedimiento anterior sobre el nuevo frente hasta haberlos generado todos.

Algorithm 3: Función de ordenación en frentes no dominados

```

1 Function ordenarPorFrentesNoDominados (población)
2    $F \leftarrow \emptyset$ 
3   /* Identificar el frente  $F_1$  */
4   foreach  $p \in \text{población}$  do
5     /* Conjunto de soluciones dominadas por  $p$  */
6      $S_p = \emptyset$ 
7     /* Números de soluciones que dominan a  $p$  */
8      $n_p = 0$ 
9     foreach  $q \in \text{población}$  do
10       if  $p \prec q$  then /*  $p$  domina a  $q$  */  

11          $S_p = S_p \cup \{q\}$ 
12       else if  $q \prec p$  then /*  $q$  domina a  $p$  */  

13          $n_p = n_p + 1$ 
14       end
15       if  $n_p = 0$  then  

16          $p_{rango} = 1$ 
17          $F_1 = F_1 \cup \{p\}$ 
18       end
19     end
20   end
21    $i = 1$ 
22   while  $F_i \neq \emptyset$  do
23      $Q = \emptyset$ 
24     foreach  $p \in F_i$  do
25       foreach  $q \in S_p$  do
26          $n_q = n_q - 1$ 
27         if  $n_q = 0$  then  

28            $q_{rango} = i + 1$ 
29            $Q = Q \cup \{q\}$ 
30         end
31       end
32     end
33      $i = i + 1$ 
34      $F_i = Q$ 
35   end
36 fin

```

Figura 2.12: Pseudocódigo de la función de ordenamiento utilizada en NSGAII [34]

Densidad de estimación (Crowding Distance)

Otra función presente en la Figura 2.9 es la asignación de las densidades sobre cada solución, la cual corresponde a la distancia promedio entre la solución anterior y la siguiente a partir de cada uno de los objetivos. En la Figura 2.13 la densidad de la solución i corresponde a la suma de la longitud del lado mayor y del lado menor del rectángulo.

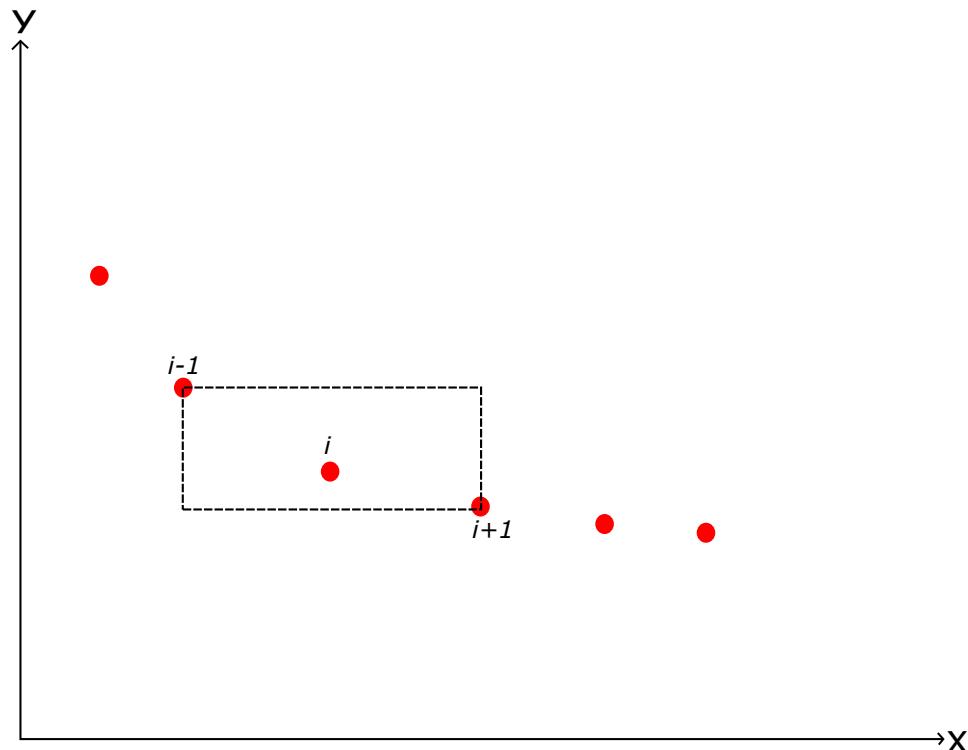


Figura 2.13: Cálculo de la densidad de estimación al rededor de la solución i [34]

El procedimiento para esta función se muestra en la Figura 2.13 y consiste en:

Primero, crear e inicializar un arreglo $\mathcal{I}_{distancia}$, con el valor 0, en donde guardar las distancias para cada solución a medida que se van calculando. Luego, por cada uno de los objetivos se ordena el frente \mathcal{I} y dentro $\mathcal{I}_{distancia}$ se le asigna al primer y ultimo elemento el valor infinito. Finalmente, se recorre desde la segunda hasta la penúltima solución calculando la distancia $\mathcal{I}[i]_{distancia}$. Notar que f_m^{max} y f_m^{min} corresponden al valor del objetivo m de la primera y última solución.

Algorithm 4: Función de calculo de densidades

```

1 Function asignarDensidad ( $\mathcal{I}$ ) /*  $\mathcal{I}$  : frente de soluciones no
   dominadas */
2    $l = |\mathcal{I}|$  /* Obtiene el tamaño del frente */ 
3
4   /* Inicializa la distancia para cada solución */ 
5   foreach  $i \leftarrow 1$  to  $l$  do
6     |  $\mathcal{I}[i]_{distancia} \leftarrow 0$ 
7   end
8   foreach  $m \leftarrow 1$  to numero de objetivos do
9     |  $\mathcal{I} \leftarrow \text{sort}(\mathcal{I}, m)$  /* Ordena por el objetivo  $m$  */ 
10    /* Asignar a la primera y última solución el valor  $\infty$  */
11    |  $\mathcal{I}[1]_{distancia} \leftarrow \mathcal{I}[l]_{distancia} \leftarrow \infty$ 
12    for  $i \leftarrow 2$  to  $l - 1$  do
13      /* Asigna la distancia a las soluciones restantes */
14      |  $\mathcal{I}[i]_{distancia} \leftarrow \mathcal{I}[i]_{distancia} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m) / (f_m^{\max} - f_m^{\min})$ 
15    end
16  end
17 fin

```

Figura 2.14: Pseudocódigo de la función de asignación de densidad [34]

Comparador de densidad (Crowing Distance comparator)

Este operador compara las soluciones basados en dos conceptos los cuales son el rango de dominación y la densidad de soluciones. Estos fueron calculados al momento de generar los frentes y asignar la densidad a las soluciones. De acuerdo a Deb [34], se define el orden dado por el operador de densidad (\prec_n) como: $i \prec_n j$, si $(i_{rango} < j_{rango}) \circ ((i_{rango} = j_{rango}) \circ (i_{distancia} > j_{distancia}))$.

2.9. Problemas de optimización en RDA

Basado en los requisitos se han seleccionado dos problemas; optimización del diseño de RDA basado en la selección del diámetro de tuberías (*Pipe Optimizing*) y la optimización del régimen de bombeo (*Pumping Schedule*). Y en esta sección se representan sus modelos matemáticos así como la codificación implementada.

2.9.1. *Pipe Optimizing*

Pipe Optimizing es un problema de diseño cuyo objetivo es minimizar el costo de inversión en la construcción de las tuberías. Para ésto, se busca aquella combinación de diámetros que disminuyan el costo de la construcción de las tuberías a la vez que se cumplen las restricción de presión mínima impuesta sobre la red. La ecuación 2.1 presentada en [2] muestra la ecuación a optimizar.

$$\text{Costo de inversión} = \sum_{i=1}^N (C_i \times D_i \times L_i) \quad (2.1)$$

En la ecuación anteriormente presentada el término C_i se refiere al costo unitario de la tubería i , el término D_i corresponde al diámetro de la tubería y finalmente L_i hace referencia su longitud. Como se mencionó anteriormente, el problema debe satisfacer la siguiente restricción:

$$H_i < H_{min} \quad (2.2)$$

donde H_i corresponde a la presión sobre la tubería i y H_{min} a la presión mínima de la red.

La solución retornada por el algoritmo utilizado, en este caso GA, se puede ver en la Figura 2.15. En dicha solución los valores que toman la variable de decisión corresponden al índice a una tabla donde se encuentra el diámetro y el costo de la tubería. El largo de la tubería esta configurado en el archivo de configuración de red (El archivo con extensión imp).

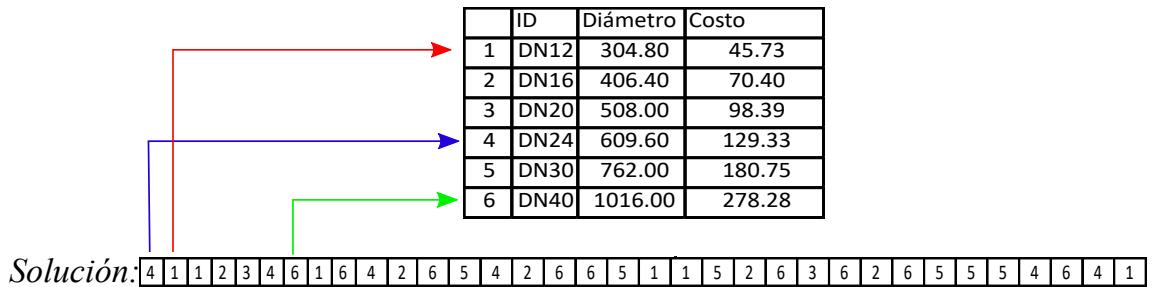


Figura 2.15: Representación de la solución del problema monoobjetivo *Pipe Optimizing*

2.9.2. Pumping Schedule

Pumping Schedule [35, 3] es un problema de operación que tiene como objetivos optimizar tanto el costo energético, así como el costo de mantenimiento de los equipos de bombeos. A continuación se expresan las ecuaciones utilizadas para calcular los objetivos y las restricciones.

Para el cálculo de los costos energéticos se ocupa la siguiente ecuación:

$$C_E(S) = \sum_{n=1}^{NP} \sum_{t=0}^{NT-1} (P_c(t) \times E_c(n, t) \times S(n, t)) \quad (2.3)$$

donde:

- $C_E(S)$: Costos energéticos.
- NP : El numero de bombas.
- NT : Número de periodos de simulación. El máximo son 24 horas.
- $P_c(t)$: La tarifa energética en el periodo t .
- $E_c(n, t)$: Consumo energético de la bomba n en el tiempo t .
- $S(n, t)$: El estado de la bomba. 1 si esta encendida y 0 si esta apagada.

Para calcular el consumo energético de la bomba n se utiliza la siguiente formula:

$$E_c(n, t) = \frac{10^{-3} \times \gamma \times Q(n, t) \times h(n, t)}{e(n, t)} \quad (2.4)$$

donde:

- γ : Peso del agua.
- $Q(n, t)$: Flujo a través de la bomba n en el tiempo t
- $h(n, t)$: Altura manométrica de la bomba.
- $e(n, t)$: Eficiencia de la bomba n en el tiempo t .

Para calcular el costo de mantenimiento se calcula el número de encendido y apagados de todas las bombas en el periodo de tiempo analizado. Matemáticamente la función para calcular dicho costo corresponde a:

$$C_N(S) = \sum_{n=1}^{NP} \sum_{t=0}^{NT-1} r_t \quad (2.5)$$

donde:

- $C_N(S)$: Costo de mantenimiento.
- r_t : Valor indicando si en el periodo t hubo un cambio de estado en la bomba desde apagado a encendido. Este valor es 1 cuando la bomba ha sido encendida.

Las funciones 2.4 y 2.5 deben cumplir las siguientes restricciones:

Conservación de la masa:

$$\sum q_{in} - q_{out} = C_j \quad (2.6)$$

donde:

- q_{in} : Flujo de entrada.
- q_{out} : Flujo de salida.
- C_j : Consumo del nodo j .

Conservación de la energía:

$$\sum h_f - \sum E_p = 0 \quad (2.7)$$

donde:

- h_f : Perdida de energía por fricción.
- E_p : Energía aportada por la bomba.

Perdida de carga por fricción:

$$h_f = \frac{10,67 \times L_q^{1,85}}{CH^{1,85} \times D^{4,87}} \quad (2.8)$$

donde:

- L_q : Largo de la tubería.
- CH : Coeficiente de Hazen-Williams.
- D : Diámetro de la tubería.

Presión mínima:

$$H_i < H_{min} \quad (2.9)$$

donde:

- H_i : Presión en el nodo i .
- H_{min} : Presión mínima.

Caudal:

$$Q_{i,t} \leq Q_i^{max} \quad (2.10)$$

donde:

- $Q_{i,t}$: Caudal del nodo i en el tiempo t .
- Q_i^{max} : Caudal máximo del nodo i .

Nivel de depósito:

$$TS_{i,NT} \geq TS_{i,0} \quad (2.11)$$

donde:

- $TS_{i,NT}$: Nivel del reservorio i en el periodo de tiempo NT .
- $TS_{i,0}$: Nivel del reservorio i en el tiempo 0.

En la Figura 2.16 muestra como se codifica la solución a este problema propuesta por [3]. Como se puede observar la solución cuenta con 24 variables de decisión correspondiente a las 24 horas del día. Cada variable es un índice a la matriz de combinaciones posibles para cada bomba. Posteriormente, se genera una matriz binaria en donde cada fila es una bomba, cada columna es el periodo y el valor es el estado de la bomba en dicho periodo. Esta matriz binaria es usada para calcular el número de cambios de estado en las bombas de la ecuación 2.5, así como para obtener el estado de la bomba en el periodo t de la ecuación 2.4 referente al termino $S(n, t)$.

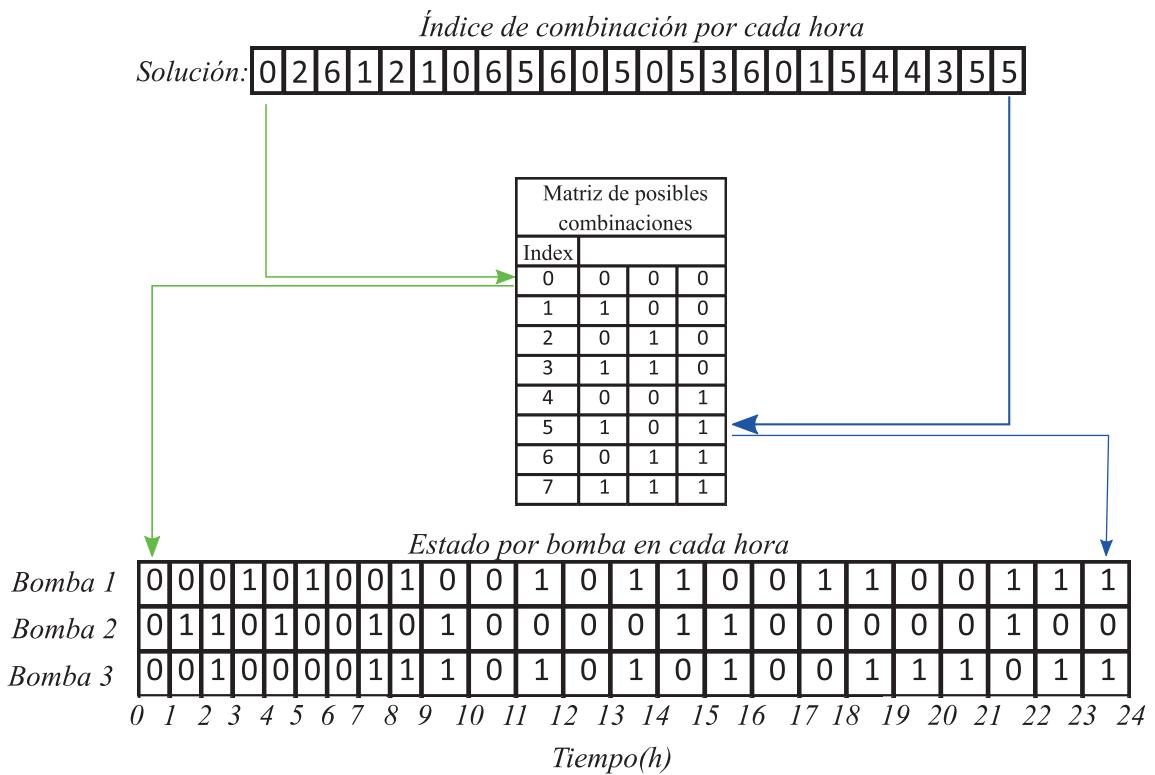


Figura 2.16: Representación de la solución problema multiobjetivo *Pumping Schedule*

2.10. Trabajo relacionado

En este apartado se mostrarán distintas herramientas y el enfoque utilizado para resolver el problema con ellas. En general, los enfoques consisten en usar software ya disponible o crear un software personalizado.

- **Magmoredes:** En [36] se describe la existencia de un software de diseño basado en micro-algoritmos genéticos multiobjetivos, que de acuerdo al autor, tiene un mejor rendimiento y es más eficiente que el algoritmo NSGAII. Esto se debe a que requiere una menor cantidad de memoria y tiene un mejor tiempo de cómputo. Este programa puede cargar cualquier red y realiza los cálculos utilizando librerías de Java. Las funciones objetivos que este sistema intenta resolver son la optimización de los costos de construcción de la red y la confiabilidad final de la red.
- **WaterGEMS:** Software comercial que permite la construcción de modelos geoespaciales; optimización de diseño, ciclos de bombeo y calibración automática del modelo; y la gestión de activos. Este software a sido usado en [37] para llevar a cabo las simulaciones necesarias para su estudio. La metodología seguida para la utilización de este sistema consiste en ingresar los datos a WaterGEMS para correr las simulaciones. La limitación de este programa es que no permite la adición de nuevos algoritmos por parte del usuario, en el caso de que se quiera probar o mejorar algún algoritmo ya existente. Sin embargo, este sistema también tiene sus ventajas, porque ya incorpora algunos algoritmos predefinidos para resolver ciertos problemas. Además, posee diversas funcionalidades como conexión con datos externos, operaciones de análisis espaciales, intercambio de datos con dispositivos o programas de administración, entre otros.
- **EPANET:** El enfoque seguido con la utilización de esta herramienta consiste en automatizar la ejecución de los algoritmos y la posterior evaluación de los resultados utilizando la librería EPANET Toolkit. Este es usado en [23] en donde se implementan ciertos algoritmos metaheurísticos y los resultados obtenidos por estos son enviados a la EPANET Toolkit para evaluar la solución y determinar la factibilidad de ésta. La ventaja de este enfoque es que permite una mayor flexibilidad en los algoritmos metaheurísticos utilizados y los problemas que se quieren resolver. Sin embargo, debido a que se necesita implementar los problemas y los algoritmos, este enfoque toma mucho tiempo.

Para el desarrollo de este proyecto se usa el enfoque basado en EPANET, puesto que es una librería de simulación hidráulica ampliamente utilizada y permite enfocarnos en la resolución de los problemas usando algoritmos metaheurísticos. Tanto

Magmoredes como WaterGEMS buscan resolver temas concretos en los sistemas de distribución de agua potable y puesto que el código de estos programas no esta disponible públicamente o son un sistema que se comercializa sin permitir la modificación del sistema por parte de terceros, se busca con nuestro proyecto incorporar esta capacidad para que en futuros trabajos se pueda abarcar una mayor cantidad de problemas en nuestro sistema.

3. Metodología de desarrollo

En este capítulo se presenta la metodología a seguir durante el desarrollo del proyecto, así como las fases que la componen y las tareas que son realizadas por cada una de estas fases.

La metodología escogida para utilizar durante el desarrollo de este proyecto es la metodología iterativa e incremental.

Debido a que la metodología esta pensada para ser llevada a cabo por un equipo de trabajo, esta se ha adaptado para poder ser aplicada en el desarrollo llevado a cabo por una sola persona. Esta adaptación puede ser resumida en los siguientes puntos:

- Disminuye la cantidad de documentación generada por cada fase.
- Permite llevar a cabo más de una fase dentro de cada iteración al mismo tiempo.
- Los roles de analista, diseñador, implementador y tester son realizados por una sola persona.

Las tareas a desempeñar por cada fase consisten en:

Análisis:

- **Captura de requisitos:** durante esta etapa se realiza una reunión con el cliente ya sea física o remotamente, se conversa y llega a un acuerdo acerca de las funcionalidades que deben ser implementadas en la aplicación.
- **Priorización de los requisitos:** junto con el interesado se le asigna una prioridad a los requisitos capturados con el fin de establecer el orden en que estos deben ser implementados.

- **Especificación formal de requisitos:** en esta etapa se redacta un documento aparte donde se especifican de manera formal los requisitos. La especificación contiene los siguientes datos por cada uno de los requisitos identificados: id del requisito, descripción, fuente, prioridad, estabilidad, fecha de actualización, estado de la implementación, incremento y el tipo de requisito.

Diseño:

- **Definición y especificación de la arquitectura del sistema:** durante esta etapa se especifica tanto la arquitectura física como lógica de la aplicación.
- **Diseño de las interfaces de usuario:** se diseñan las interfaces de usuario de la aplicación. Este diseño se realiza utilizando herramientas de *mockup*, implementando las interfaces directamente sin funcionalidad.
- **Diseño de los componentes:** se realiza un diagrama de los componentes que conforman la aplicación.
- **Especificación formal del diseño del sistema:** se redacta un documento en el que se presentan los diagramas, las interfaces y los detalles de la implementación.

Implementación:

- **Programación de los componentes de software:** implementación de las funcionalidades para cumplir con los requisitos.
- **Integración de los componentes de software:** consiste en tomar cada uno de los módulos independientes e integrarlos dentro de un solo sistema.
- **Elaboración del producto entregable:** se empaqueta la aplicación para generar un archivo que pueda ser distribuido. Éste, puede ser un archivo con extensión .jar o generar una aplicación que posea todas las dependencias, incluyendo Java, y que se ejecuta a partir del archivo de extensión exe.
- **Elaboración de manual de usuario:** consiste en redactar un manual de usuario que indica como realizar las tareas dentro de la aplicación.

Pruebas:

- **Definición de casos de prueba:** esta tarea consiste en identificar las pruebas que deben ser realizadas para validar los componentes del sistema. Se dividen en dos tipos de prueba; automatizadas y manuales.
- **Especificación formal de pruebas:** se redacta un documento en donde se especifican las pruebas realizadas. Para cada especificación de los caso de prueba se definen los siguientes elementos: id de prueba, título, característica a evaluar, el objetivo de la evaluación, la configuración de la aplicación al momento de realizar la prueba, los datos de prueba, las acciones que hay que realizar durante la prueba y finalmente los resultados esperados de la ejecución.
- **Ejecución de pruebas:** durante esta etapa se ejecutan las pruebas sobre el programa y se documenta los errores encontrados para ser resueltos en la iteración posterior, si es que son complejos de resolver. Si los errores encontrados son simples se resuelven en el momento.

Cada una de las fases mencionadas anteriormente tiene como resultado un documento de especificación formal con el siguiente contenido:

Especificación formal de requisitos:

- **Introducción:** En este apartado del documento se da una introducción al problema que se ha identificado.
- **Requisitos de usuario:** Consiste en la recopilación de lo requisitos de los usuarios que deben ser cumplidos al final del periodo de desarrollo.
- **Requisito de sistema:** Son los requisitos, desde un punto de vista mas técnico, que son necesarios para satisfacer los requisitos de usuario.
- **Matriz de trazado requisitos de usuario versus sistema:** Matriz que permite ver la trazabilidad de los requisitos de usuario con los de sistema.

Especificación formal del diseño del sistema:

- **Casos de uso:** Serie de diagramas que permiten ver la interacción que el usuario tiene con el sistema.

- **Arquitectura física:** Descripción de los componentes físicos que intervienen en la aplicación.
- **Arquitectura lógica:** Descripción a alto nivel del software y los componentes que lo componen.
- **Diagrama de componentes:** Permite ver la división del sistema y la interacción entre los distintos componentes [38].
- **Diseño de interfaces:** Bosquejos o implementaciones de las interfaces a ser utilizadas en la propuesta.
- **Diagrama de clases:** Describe la relación entre las distintas clases presentes en la solución propuesta.

Manual de usuario: Explicación acerca de las capacidades de la aplicación, acompañada de esquemas y ejemplos de uso.

Especificación formal de pruebas: Se documenta las pruebas automatizadas y manuales que se realizan y sobre qué elemento se llevan a cabo.

La razón por la que se utiliza esta metodología sobre otras es porque el producto resultante de este proyecto está pensado para servir como base para futuros trabajos. Debido a esto, es necesario documentar detalladamente la implementación para que otros programadores puedan continuar con su desarrollo en el futuro. Aunque existen otras metodologías como Cascada u otras tradicionales, estas son difíciles de llevar a cabo por la cantidad de documentación que se requiere, mientras que metodologías de desarrollo ágil carecen en cuanto a la documentación que se necesita para el sistema a desarrollar. Adicionalmente, esta metodología nos permite obtener una retroalimentación al final de cada iteración, obtener nuevos requisitos que no hayan quedado definidos en etapas anteriores o refinar los requisitos y el diseño ya existente, permitiendo así mejorar la calidad del producto final.

La implementación de esta metodología para el desarrollo del proyecto se lleva a cabo repartiendo las tareas necesarias para el cumplimiento de los objetivos en iteraciones. De este modo al final de cada iteración se cuenta con un prototipo funcional de la aplicación sobre el que se agrega las nuevas funcionalidades en las iteraciones siguientes.

4. Desarrollo

En este capítulo se da a conocer la concepción del proyecto, la planificación de las iteraciones y se detalla como se aplica la metodología en el desarrollo del proyecto. Para esto, se presenta por cada una de las fases del desarrollo las actividades realizadas en cada iteración utilizando una serie de casos de prueba a modo de ejemplos de las funcionalidades a implementar.

4.1. Concepción del proyecto

Este proyecto se origina como una propuesta por parte del profesor del departamento de Ingeniería en Obras Civiles, Daniel Mora Melia. Él, junto a un grupo de expertos de diversas áreas, presentaron y publicaron un artículo del proyecto JHawanet [3]. Dicho artículo, presenta la integración de dos librerías independientes, JMetal y Epanet, como herramienta para llevar a cabo optimizaciones sobre RDA. JMetal [4] es un Framework de Java, orientado a la optimización multiobjetivo y es usado como motor de optimización. Mientras que Epanet [21] es una herramienta la cual permite realizar simulaciones en redes de agua potable.

Debido al artículo anteriormente mencionado, surgió la idea de crear una herramienta gráfica con el fin de facilitar la optimización de redes de agua potable. Puesto que la utilización de la herramienta JHawanet requiere conocimiento computacional avanzado. Y de esta forma, permitir el trabajo de Ingenieros hidráulicos en un entorno especialmente diseñado para su área sin perder la capacidad que la herramienta posee para que usuarios avanzados puedan incorporar y evaluar nuevos problemas y algoritmos.

4.2. Funcionalidades

En esta sección se presentan las funcionalidades que se utilizan para demostrar la aplicación de la metodología.

Se escogen 3 funcionalidades las cuales corresponden a:

Funcionalidad 1: El sistema debe poder visualizar la red. Esta funcionalidad consiste en visualizar gráficamente la forma de la red cargada. El archivo de descripción de red debe ser creado desde la aplicación Epanet y tener la extensión inp.

Funcionalidad 2: El sistema debe poder encontrar soluciones al problema de la optimización del diseño de RDA basado en la selección de diámetros de tuberías utilizando un Algoritmo Genético.

Funcionalidad 3: El sistema debe poder realizar una simulación hidráulica utilizando los valores por defecto del archivo de red.

4.3. Planificación

Como se mencionó en el capítulo anterior, para llevar a cabo el desarrollo del proyecto se optó por la metodología iterativa e incremental. La planificación resultante seguida durante el desarrollo del proyecto se muestra en el Cuadro 4.1.

Cuadro 4.1: Planificación de las iteraciones

Nº Iteración	Tareas	Fecha Inicio	Fecha término
1	- Especificación de requisitos - Escoger arquitectura lógica y física	26/08/2019	14/10/2019
2	- Implementar problema monoobjetivo (Pipe Optimizing) - Implementar Algoritmo Genético - Implementar operadores de selección y reproducción	14/10/2019	11/11/2019
3	- Crear interfaces de usuario - Guardar soluciones	11/11/2019	20/01/2020
4	- Implementar problema multiobjetivo (Pumping Schedule) - Implementar algoritmo NSGAII	27/01/2020	24/02/2020
5	- Permitir realizar múltiples repeticiones de un algoritmo - Permitir realizar la simulación usando los valores por defecto de la red	02/03/2020	04/05/2020
6	- Agregar símbolos al dibujo de la red - Exportar a excel - Agregar menu de configuración	11/05/2020	08/06/2020

4.4. Requisitos

Durante la fase de requisitos se llevo a cabo la captura, priorización y la especificación formal de requisitos.

Los requisitos iniciales de la aplicación fueron capturados a partir de una reunión con el profesor Jimmy Gutierrez. Posteriormente, estos requisitos fueron priorizados para finalmente ser documentados en el documento de especificación formal de requisitos.

A medida que avanzaban las iteraciones algunos requisitos fueron cambiando o fueron surgiendo requisitos nuevos. En el Cuadro 4.2 se detallan los cambios y actividades realizados durante cada iteración.

Cuadro 4.2: Actividades y cambios de la fase de requisitos durante cada iteración

Nº Iteración	Requisitos cubiertos	Tareas
1	5/32	Se capturan 5 requisitos. Se crea el informe de especificación de requisitos.
2	11/32	Se capturan 6 nuevos requisitos. Se actualiza el documento de requisitos.
3	16/32	Se capturan 5 nuevos requisitos. Se actualiza el documento de requisitos.
4	20/32	Se capturan 4 nuevos requisitos. Se actualiza el documento de requisitos.
5	20/32	No hay nuevos requisitos.
6	32/32	Se capturan 12 nuevos requisitos. Se actualiza el documento de requisitos.

Los cuadros 4.3, 4.4 y 4.5 muestran la especificación formal de los requisitos relacionados a las funcionalidades escogidas anteriormente.

Cuadro 4.3: Especificación del requisito de usuario RU004.

RU004 – Visualizar red en una interfaz gráfica.	
Descripción:	Se debe mostrar en la interfaz gráfica una representación de la red (Un dibujo, etc) generada a partir de la información contenida en el archivo inp.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

Cuadro 4.4: Especificación del requisito de usuario RU002.

RU002 – Resolver el problema monoobjetivo (<i>Pipe Optimizing</i>) usando un Algoritmo Genético.	
Descripción:	El Algoritmo Genético debe ser aplicado para resolver el problema monoobjetivo que tiene como función objetivo el costo de inversión y como variable de decisión el diámetro de las tuberías.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

Cuadro 4.5: Especificación del requisito de usuario RU020.

RU020 – Permitir realizar simulaciones hidráulicas utilizando los valores por defectos que vienen en el archivo inp y visualizar los resultados.	
Descripción:	Utilizando los valores que vienen por defecto en el archivo inp se debe poder llevar a cabo la simulación hidráulica de la red. Posteriormente, los resultados podrán ser visualizados por el usuario.
Fuente:	Daniel Mora-Meliá
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

La especificación formal de los requisitos restantes se encuentra en el **Anexo A**.

4.5. Diseño

Una vez terminada la fase de requisitos se procede a diseñar la aplicación. Dentro del diseño, se encuentran las tareas de escoger y documentar la arquitectura física y lógica, realizar los diagramas de clases, diagramas de secuencia, diseño de interfaces, entre otros.

Mientras avanzaba el desarrollo fue necesario ir modificando el documento de diseño debido a la aparición o al cambio de requisitos, así como la realización de mejoras en los diagramas realizados. En el Cuadro 4.6 se presentan más detalladamente los cambios realizados en cada iteración.

Cuadro 4.6: Actividades y cambios en la fase de diseño durante cada iteración

Nº Iteración	Tareas	Comentario
1	Crear arquitectura lógica. Crear arquitectura física. Diseñar módulos. Crear diagrama de clases del módulo metaheurística. Crear diagrama de clases de la representación de la red.	Durante esta iteración se creó el documento de diseño y se crearon los esquemas básicos para orientar la construcción del software.
2	No hubieron cambios	No hubieron cambios en esta iteración en el aspecto del diseño.
3	Diseñar interfaces de usuario. Especificar detalles de la implementación referente a las anotaciones de Java (<i>Java Annotations</i>) y <i>Java Reflection</i> . Generar diagrama de secuencia de optimización.	Durante esta fase de diseño se desarrollaron algunas de las interfaces de usuario de la aplicación. Los diagramas de secuencia creados indican la interacción entre las clases para poder realizar una tarea.
4	No hubieron cambios	No hubieron cambios en esta iteración en el aspecto del diseño.
5	Modificar interfaces de usuario. Crear diagrama de secuencia para realizar la simulación usando las configuraciones por defecto. Modificar y mejora del diagrama de secuencia de la optimización.	Debido a requisitos del usuario en el área de usabilidad de la aplicación fue necesario modificar las interfaces.
6	Modificar y mejora del diagrama de secuencia de la optimización.	Se modificó el diagrama de secuencia de la optimización debido a la aparición de un nuevo requisito.

Como una de las primeras tareas realizadas durante esta fase se procedió a definir la arquitectura de la aplicación, tanto desde el punto físico como el lógico.

La arquitectura física del programa a desarrollar es la arquitectura monolítica, cuya característica es que el software que posee esta arquitectura funciona localmente sin necesidad de interactuar con otros equipos. Esta elección se debe a que en reuniones con el cliente se estableció que el programa será usado localmente y debe operar sobre el sistema operativo Windows. Limitación debida principalmente al acceso a funciones nativas cuando se realizan simulaciones hidráulicas utilizando la librería dinámica de Epanet. La Figura 4.1 muestra el diagrama de la arquitectura monolítica.

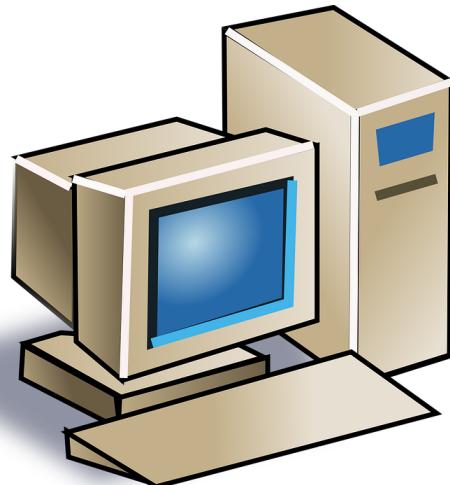


Figura 4.1: Arquitectura física monolítica

En cuanto a la arquitectura lógica se escogió utilizar Modelo-Vista-Controlador. Ésta elección es debido al hecho de que dicha arquitectura nos permite separar la capa de interfaz de usuario de la lógica de la aplicación mejorando la escalabilidad y mantenibilidad del software. La Figura 4.2 muestra el diagrama de la arquitectura lógica, así como los principales módulos de la aplicación.

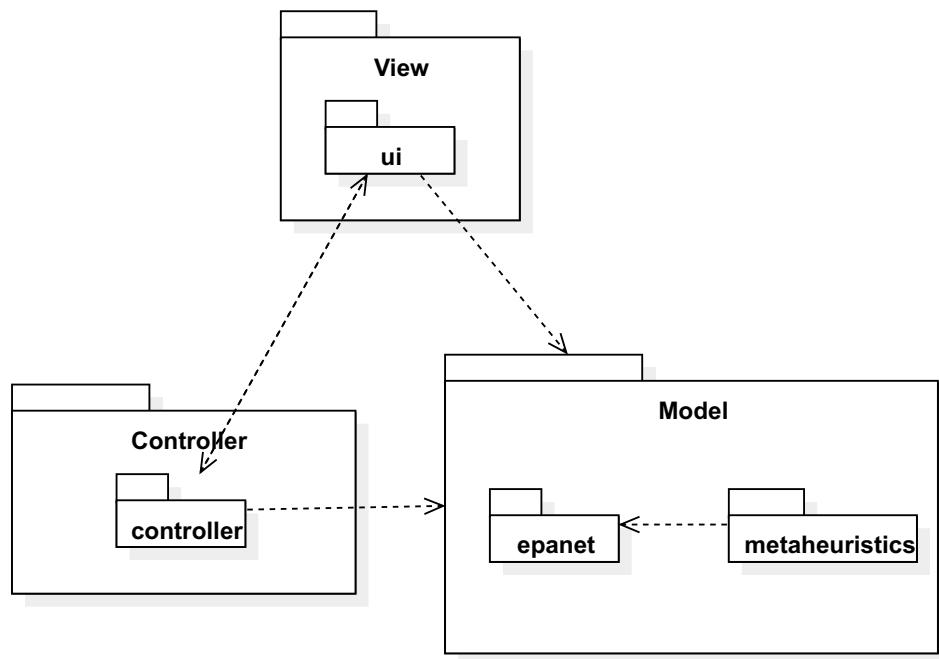


Figura 4.2: Arquitectura lógica Modelo-Vista-Controlador

A continuación se presentarán los diseños relacionados por cada una de las funcionalidades escogidas con anterioridad.

Funcionalidad 1: Para implementar esta funcionalidad es necesario abstraer en un conjunto de clases los datos almacenados en el archivo de configuración de red. La Figura 4.3 muestra el diagrama de clases resumido (se dejan fuera varias clases que especifican las configuraciones generales de la red) de la estructura sobre la que se almacenan los datos al momento de cargar la red. La clase *Network* actúa como un contenedor para las demás clases presentes en el diagrama.

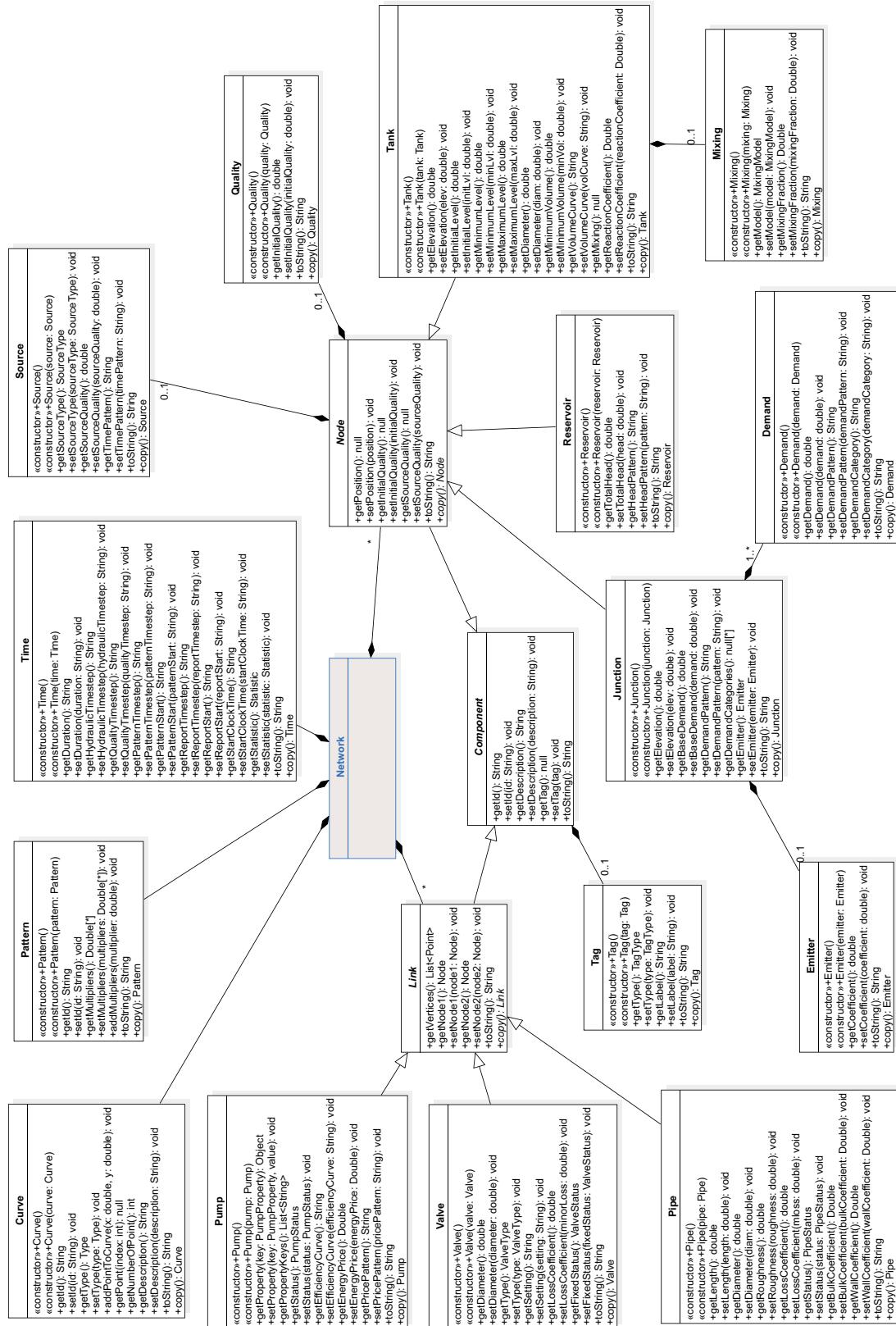


Figura 4.3: Diagrama de clases de la abstracción de la red resumido

Funcionalidad 2: Esta es una de las funcionalidades ligada al modulo metaheurística y debido a uno de los requisitos establecidos por el cliente de que la aplicación debe poder extender el número de algoritmos, operadores y problemas implementados es necesario implementar una jerarquía de clases que facilite lo anteriormente mencionado. Es por ello, que se utilizó como base la jerarquía de clases utilizada por el Framework JMetal [4] la cual fue adaptada para ser ocupada por esta aplicación. La Figura 4.4 corresponde a la adaptación mencionada anteriormente.

Debido a la importancia que tienen las metaheurísticas para la aplicación, se explicará de manera general que es lo que realiza cada una de las clases e interfaces presentes en la Figura 4.4.

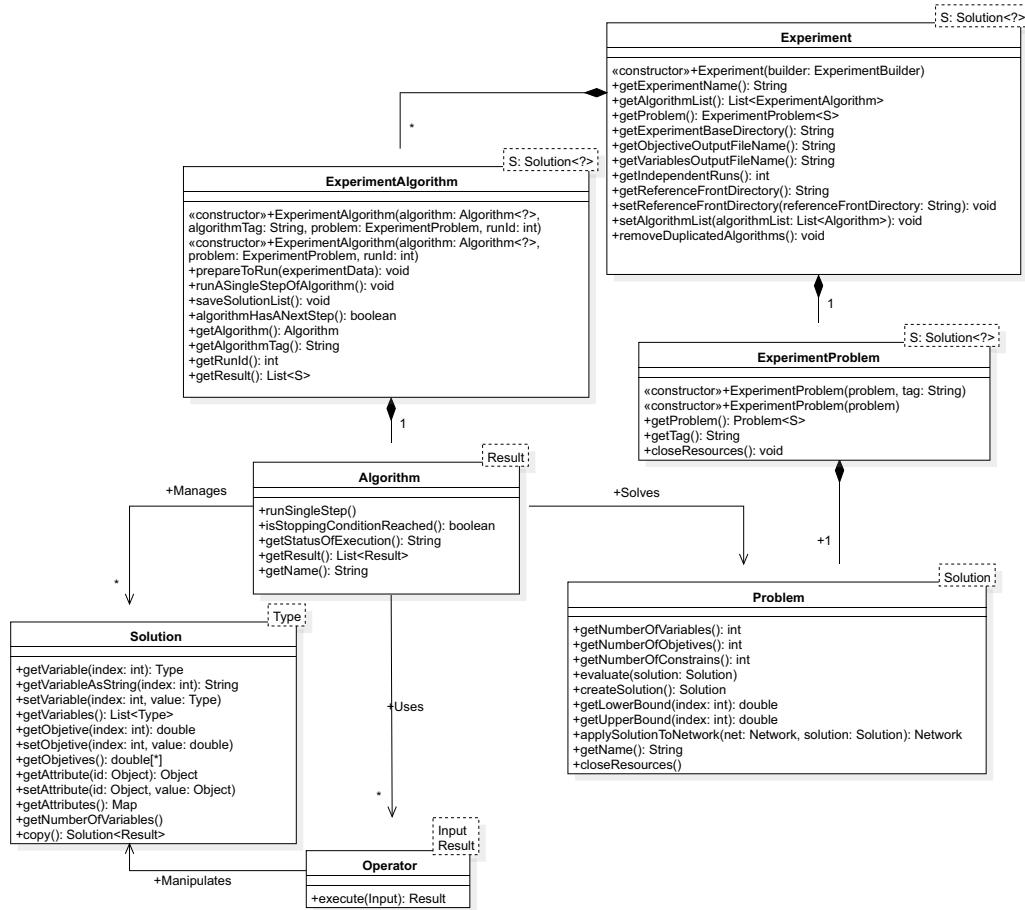


Figura 4.4: Diagrama de clases modulo metaheurística. Modificación a partir del diagrama presentado en [4]

Algorithm es la interfaz desde la que se heredan, ya sea directa o indirectamen-

te, cada uno de los algoritmos metaheurísticos implementados por la aplicación. Su principal método es *runASingleStep* puesto que éste permite realizar una única iteración del algoritmo, permitiendo así que el programa interactúe con la instancia del algoritmo. Esta interacción permite recuperar los resultados intermedios de la optimización, así como también cancelar la ejecución. Un ejemplo de la implementación de este método para GA se muestra en la Figura 4.5.

```

int step = 0;
List<IntegerSolution> population;
public void runSingleStep() throws Exception, EpanetException {
    List<S> offspringPopulation;
    List<S> selectionPopulation;

    // Durante la primera iteración inicializa
    // y evalúa la población
    if (step == 0) {
        population = createInitialPopulation();
        population = evaluatePopulation(population);
        initProgress();
    }
    // Desde la segunda iteración en adelante selecciona
    // y reproduce las soluciones
    if (!isStoppingConditionReached()) {
        selectionPopulation = selection(population);
        offspringPopulation = reproduction(selectionPopulation);
        offspringPopulation = evaluatePopulation(offspringPopulation);
        population = replacement(population, offspringPopulation);
        updateProgress();
    }

    this.step++;
}

```

Figura 4.5: Código del método *runSingleStep* utilizado con GA y NSGAII

Solution es la interfaz que representa la solución del problema. La aplicación implementa esta interfaz en una clase llamada *IntegerSolution*, la cual permite trabajar variables de decisión de tipo entero.

Operator es la interfaz de la que heredan todos los tipos de operadores usados en la aplicación. Los operadores son clases que contienen una función que trabaja sobre una o un conjunto de soluciones. Sus usos son muy variables, entre ellos se encuentra la selección de soluciones dentro de un conjunto, la modificación de las variables de decisión de una solución y la combinación de dos soluciones con el fin

de generar unas nuevas.

Problem es la interfaz de la que heredan todos los problemas. Las clases que implementan esta interfaz son las que se encargan de evaluar y penalizar las soluciones generadas por los algoritmos metaheurísticos en el método *evaluate*. Adicionalmente, esta clase es usada para mapear una solución sobre la red cargada y de esta manera poder generar un nuevo archivo de configuración de red. El método que realiza esto ultimo es *applySolutionToNetwork*.

Las clases **ExperimentAlgorithm** y **ExperimentProblem** actúan como envoltorios para las clases *Algorithm* y *Problem* incorporando algunas funciones adicionales.

Por ultimo la clase **Experiment** es un contenedor de algoritmos. Cada algoritmo agregado al experimento corresponde a una iteración independiente. Actualmente, se pensaron los experimentos para solo contener varias instancias del mismo algoritmo. Los algoritmos deben ser acoplados a un *ExperimentAlgorithm* antes de ser agregados al Experimento.

Cada una de las clases explicadas anteriormente tiene un bajo acoplamiento entre ellas, así como del resto de la aplicación puesto que se hace uso de la técnica llamada Polimorfismo. Esto permite combinar cada una de ellas de diferentes maneras, por ejemplo, se pueden usar distintos algoritmos para el mismo tipo de problema. La limitación de esto es que el problema debe ser compatible con el algoritmo, es decir, si el algoritmo esta diseñado para ser monoobjetivo, entonces el problema debe tener un solo objetivo.

Funcionalidad 3: Para implementar esta funcionalidad es necesario implementar un conjunto de clases las cuales guardarán los resultados de la simulación para cada uno de los nodos y enlaces de una red. Estas nuevas clases son las mostradas en la Figura 4.6.

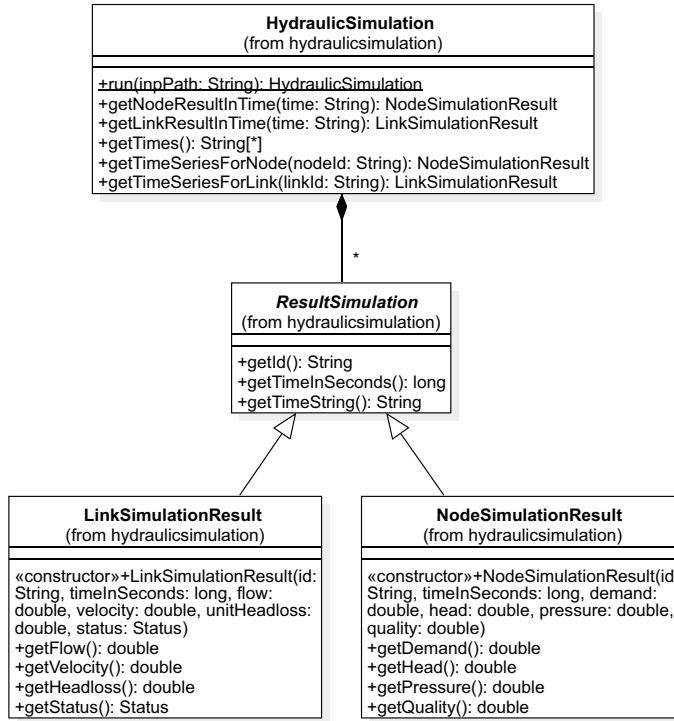


Figura 4.6: Diagrama de clases para la simulación hidráulica

4.5.1. Detalles de implementación

En diferentes secciones de este documento se ha mencionado que la aplicación debe ser extensible, refiriéndonos al hecho de que se deben poder agregar nuevos algoritmos, operadores y problemas. Una vez alcanzada la tercera iteración fue necesario hacer frente a este problema. Sin embargo, éste no es un problema fácil de abordar. Las ideas que surgieron para tratar este problema eran diversas. Una de ellas, por ejemplo, consistía en que el usuario una vez haya creado un nuevo problema; implementara por si mismo la interfaz de configuración para éste y la integrara a la aplicación. No obstante, realizar ésta tarea consistía en que se debía tener conocimiento de la implementación de interfaces gráficas usando JavaFX, así como conocer el código de la aplicación para realizar cambios sobre éste.

Finalmente se opta por utilizar un enfoque parecido al de los frameworks de Java Spring y JUnit utilizando las técnicas o patrones de diseño de inyección de dependencias e inversión de control. Para ello, se define un flujo de trabajo general para la aplicación. El flujo que sigue la aplicación se presenta en la Figura 4.8.

Se puede agregar un nuevo problema definiendo una nueva clase que es usada como plantilla para crear los nuevos experimentos con un algoritmo específico. Esta nueva clase debe heredar de una de las subinterfaces de *Registrable*. La jerarquía de herencia de la clase *Registrable* se muestra en la Figura 4.7. En dicha clase se pueden usar las anotaciones definidas para la aplicación en el constructor para especificar los tipos de las instancias a injectar, así como los valores de configuración que necesita el experimento. En este caso, las instancias inyectadas corresponden a los operadores a utilizar en el algoritmo del experimento a configurar.

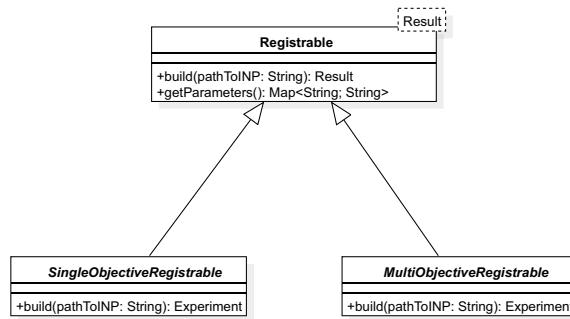


Figura 4.7: Interfaz de clase *Registrable*.

Durante la actividad de “Configurar Experimento” que se muestra en la Figura 4.8, se leen las anotaciones para construir la interfaz gráfica, una vez configurado el experimento en esta interfaz se crean e inyectan las dependencias al momento de crear la instancia de *Registrable* usando la *Java Reflection API*.

Las anotaciones permitidas en la interfaz *Registrable* que se definieron para la herramienta son las siguientes:

@NewProblem: Esta anotación permite indicar el nombre del problema y el algoritmo utilizado para resolverlo.

@Parameters: Esta anotación permite agregar información acerca de los parámetros recibidos por el constructor. Como argumento para esta anotación se pueden usar otras las cuales son:

@OperatorInput: Indica que se espera recibir un operador. Se utiliza *@OperatorOption* para indicar los posibles operadores que pueden ser inyectados.

@OperatorOption: Indica un posible operador para ser utilizado dentro del algoritmo.

@FileInput: Indica que se espera recibir una ruta a un archivo. El archivo recibido puede contener valores para configurar el algoritmo.

@NumberInput: Indica que se espera recibir un número (el tipo de este número lo indica el parametro en el constructor). Este número puede ser por ejemplo el número de iteraciones del algoritmo, etc.

@NumberToggleInput: Permite crear grupos de entradas numéricas excluyentes entre si. Por ejemplo, si solo se puede configurar uno de los dos parámetros, ya sea el número de generaciones o el número de iteraciones sin mejora.

Estas anotaciones deben estar sobre el constructor publico de la clase *Registrable*.

Los operadores también pueden hacer uso de anotaciones. Las anotaciones permitidas para los operadores es:

@DefaultConstructor: Esta anotación indica el constructor por defecto del operador. El constructor que hace uso de esta anotación es el utilizado para instanciar al operador. Dentro de esta anotación se puede usar **@NumberInput** para definir los parámetros que pueden ser inyectados.

La aplicación leyendo las anotaciones presentadas anteriormente, construye una interfaz de usuario que se abre al momento de querer resolver el problema registrado.

En el **Anexo B** se puede ver el documento de diseño de la aplicación. En dicho documento se explican con más detalle las anotaciones definidas.

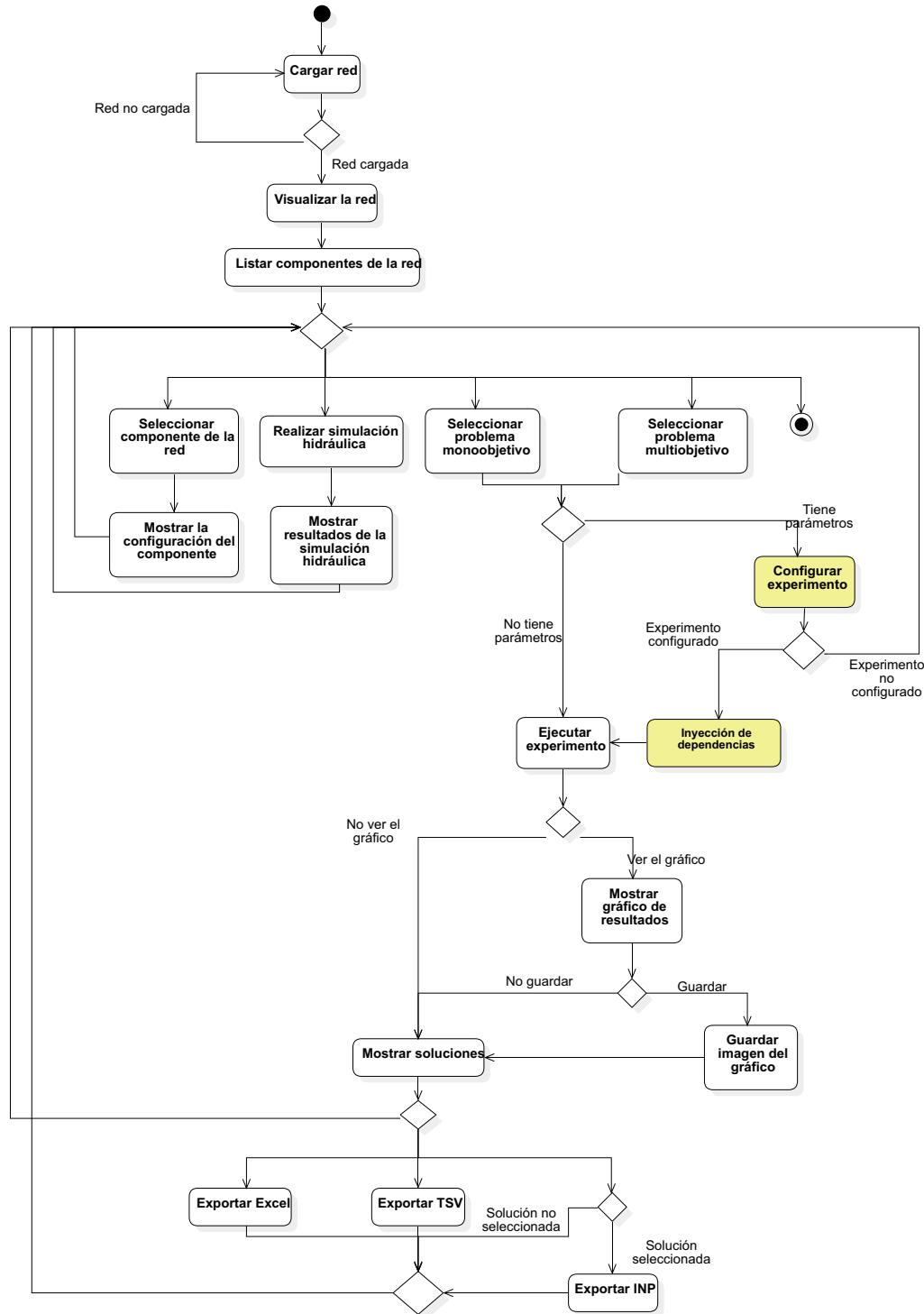


Figura 4.8: Diagrama de actividades de la aplicación

4.6. Implementación

Con la fase de diseño finalizada, se procede a realizar la fase de implementación. Durante esta fase se genera el código de la aplicación y se genera el manual de usuario.

La aplicación fue desarrollada en el lenguaje Java version 1.8. Con el fin de crear las interfaces de usuario se utilizo el Framework JavaFX.

En el cuadro 4.7 se presenta las tareas realizadas durante esta fase del desarrollo del software.

Cuadro 4.7: Actividades fase de implementación

Nº Iteración	Descripción
1	Se codifican las clases para cargar la red <i>Network</i> .
2	Se codifican las clases del modulo metaheuristica. Se implementa el Algoritmo Genético. Se implementa la codificación del problema monoobjetivo <i>Pipe Optimizing</i> . Se implementan los operadores IntegerRandomMutation, SBXCrossover, IntegerPolynomialMutation, IntegerRangeRandomMutation y UniformSelection.
3	Se implementan la interfaz de usuario principal. Se implementa el componente para visualizar la red Se implementa la interfaz de configuración del problema. Se implementa la interfaz de visualización de resultados de optimización. Se implementa la interfaz que muestra el gráfico con los resultados de la optimizacion. Se implementa la funcionalidad para guardar las soluciones en TSV. Se implementa funcionalidad para exportar la solución escogida como un inp(Formato del archivo de configuración de red).

4	<p>Se agrega el algoritmo NSGAII.</p> <p>Se implementa la codificación del problema multiobjetivo Pumping Schedule.</p>
5	<p>Se modifican las clases y archivos relacionados a la interfaces de usuario.</p> <p>Se implementa la funcionalidad para realizar múltiples simulaciones independientes de un mismo algoritmo para los problemas multiobjetivos (Experimentos).</p> <p>Se implementa la funcionalidad para realizar la simulación usando los valores del archivo de red.</p>
6	<p>Se modifica el componente de visualización de red para mostrar para cada tipo de elemento que conforma la red un símbolo distinto.</p> <p>Se implementa funcionalidad para mostrar una leyenda de los símbolos.</p> <p>Se implementa ventana de configuración de la aplicación.</p> <p>Se implementa la funcionalidad para realizar múltiples simulaciones independientes para los problemas monoobjetivo (Se adapta para utilizar los Experimentos antes utilizados solo en problemas multiobjetivos).</p> <p>Se permite agregar valores por defecto en la ventana de configuración del problemas.</p> <p>Se implementa la funcionalidad para exportar resultados a un Excel.</p>

A continuación se presentara para cada una de las funcionalidades escogidas las interfaces implementadas.

Funcionalidad 1 : La Figura 4.9 muestra la ventana de visualización de la red.

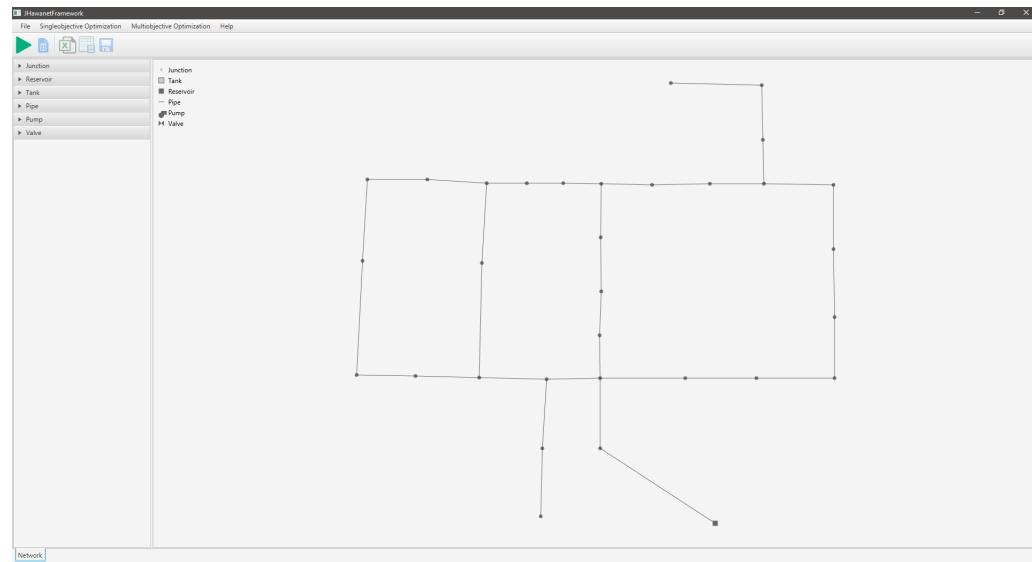


Figura 4.9: Ventana principal de la aplicación donde se visualiza la red

Funcionalidad 2 : Las Figuras 4.10, 4.11, 4.12, 4.13 y 4.14 muestran las ventanas utilizadas durante el cumplimiento de esta funcionalidad.

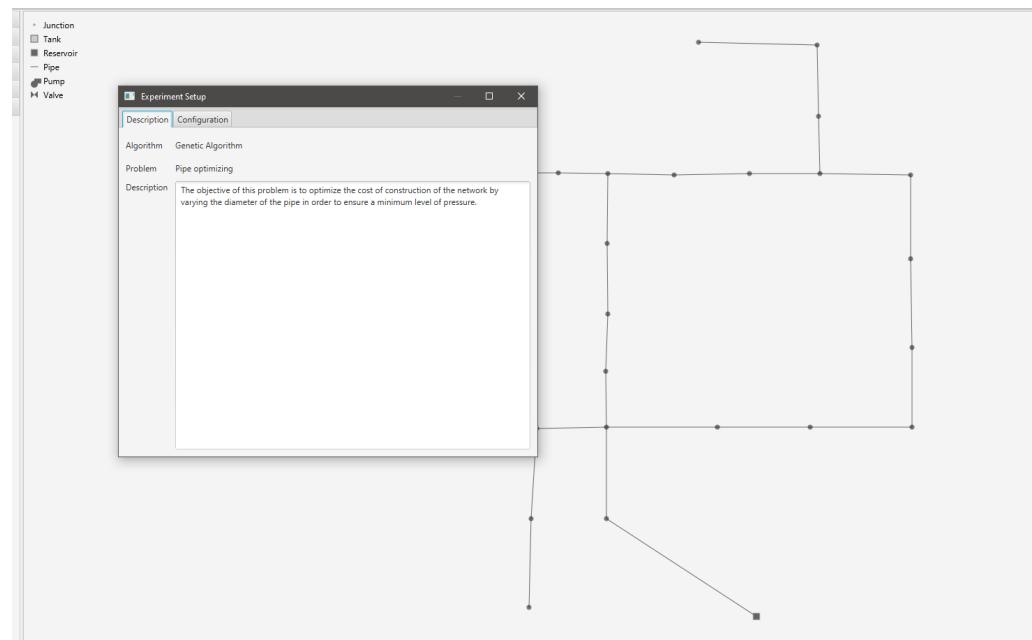


Figura 4.10: Ventana de descripción del problema.

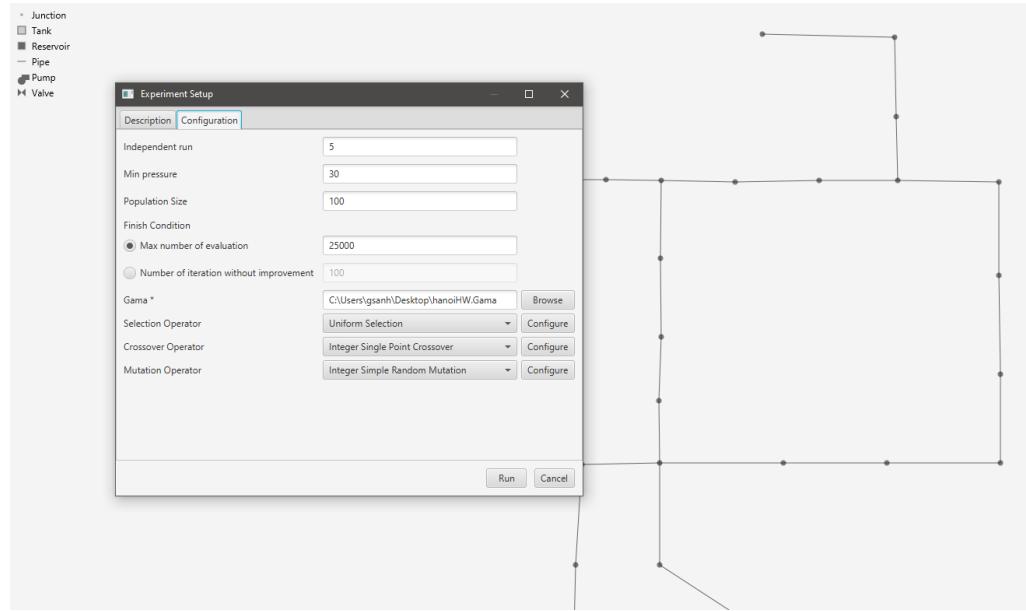
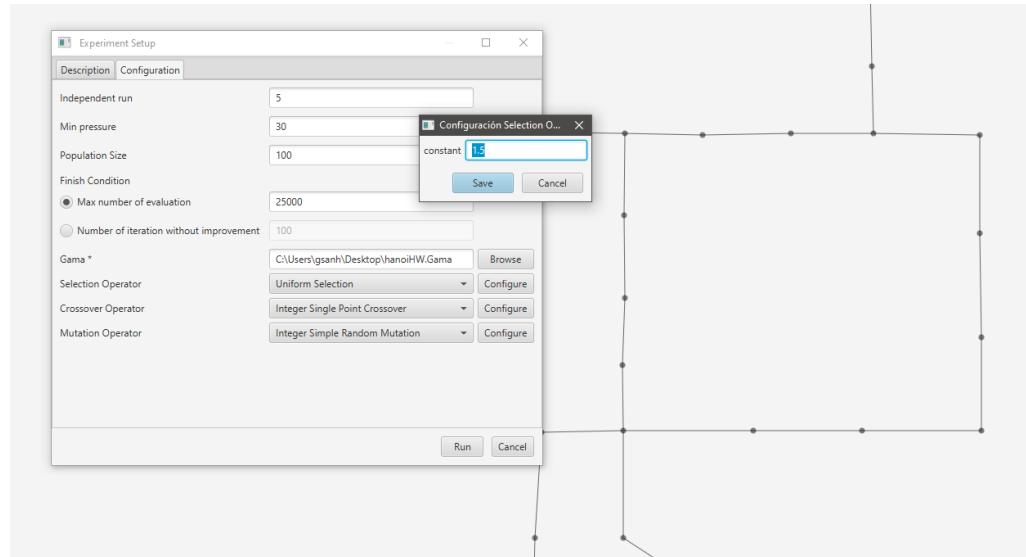


Figura 4.11: Ventana de configuración del problema.

Figura 4.12: Ventana de configuración de parámetros del operador *UniformSelection*

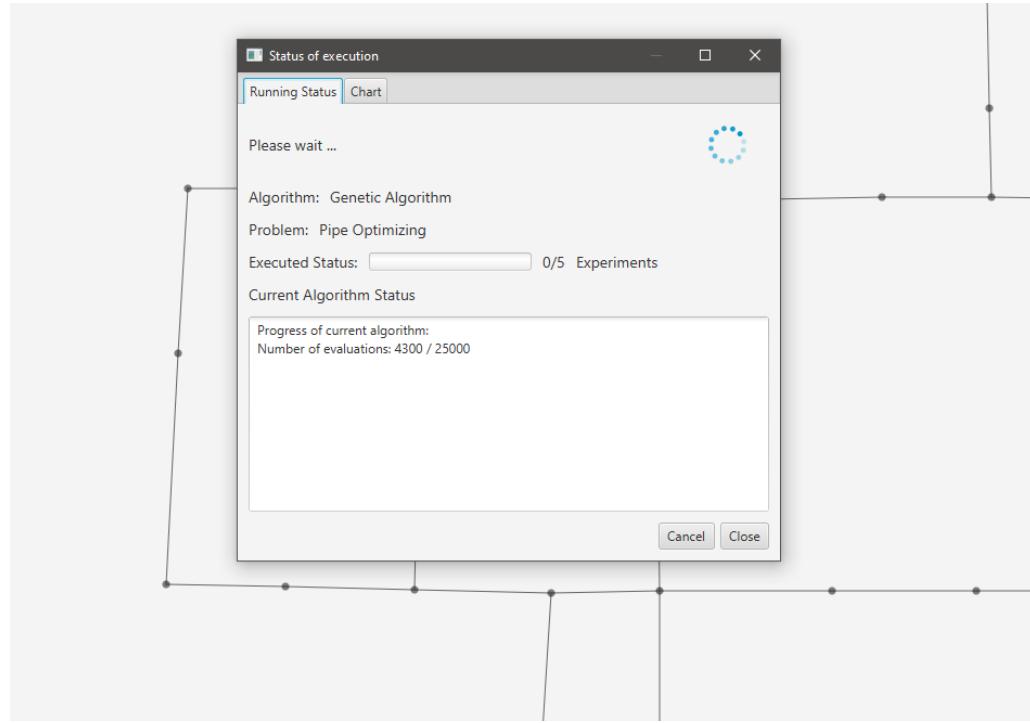


Figura 4.13: Ventana del retroalimentación mostrada durante la ejecución

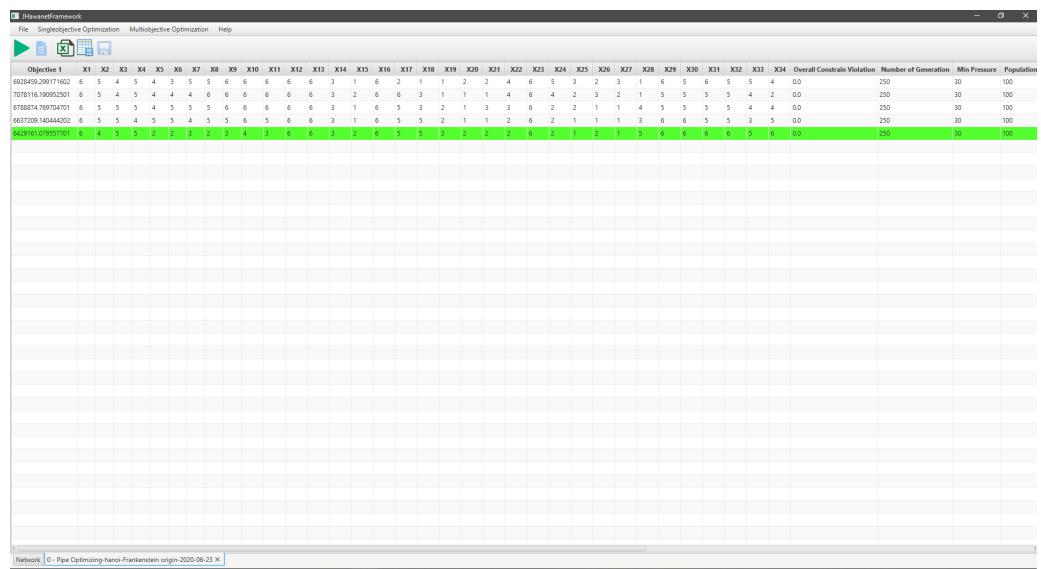


Figura 4.14: Ventana de resultados generada cuando termina la ejecución para el problema monoobjetivo Pipe Optimizing

Funcionalidad 3 : La Figura 4.15 muestra la ventana con los resultados de una simulación hidráulica para la red cargada.

The screenshot shows a software interface titled "Result of execution". A dialog box is open, prompting the user to "Select the type of table to create". Two options are available: "Network nodes at" (selected) and "Network link at". A dropdown menu shows "00:00:0". Below the dialog is a table with the following data:

Node ID	Demand	Head	Pressure	Quality
N10	145.8332977294922	50.64276123046875	50.64276123046875	0.0
N11	138.88890075683594	50.25870895385742	50.25870895385742	0.0
N12	155.55560302734375	49.97396469116211	49.97396469116211	0.0
N13	261.1111145019531	49.624454498291016	49.624454498291016	0.0
N14	170.8332977294922	50.72154235839844	50.72154235839844	0.0
N15	77.77777862548828	50.84730529785156	50.84730529785156	0.0
N16	86.1111068725586	51.03553009033203	51.03553009033203	0.0
N17	240.27780151367188	54.605525970458984	54.605525970458984	0.0
N18	373.6111145019531	57.96050262451172	57.96050262451172	0.0
N19	16.666669845581055	60.41902542114258	60.41902542114258	0.0
N2	247.22219848632812	97.14077758789062	97.14077758789062	0.0
N20	354.16668701171875	54.26154708862305	54.26154708862305	0.0
N21	258.33331298828125	53.94206619262695	53.94206619262695	0.0
N22	134.72219848632812	53.927406311035156	53.927406311035156	0.0
N23	290.2778015136719	51.09099578857422	51.09099578857422	0.0
N24	227.77780151367188	50.821048736572266	50.821048736572266	0.0
N25	47.22222137451172	50.761329650878906	50.761329650878906	0.0
N26	250.0	50.77581787109375	50.77581787109375	0.0
N27	102.77780151367188	50.8275032043457	50.8275032043457	0.0
N28	80.55555725097656	50.88719177246094	50.88719177246094	0.0
N29	100.0	50.732093811035156	50.732093811035156	0.0

Figura 4.15: Ventana de simulación hidráulica utilizando los valores del archivo de red

A continuación se muestra el constructor de utilizado en el *template* para el problema *Pipe Optimizing*. Este constructor, el cual se muestra en la Figura 4.16, genera la interfaz mostrada en la Figura 4.10 y 4.11.

```

public final class PipeOptimizingRegister implements SingleObjectiveRegistrable {
    @NewProblem displayName = "Pipe optimizing", algorithmName = "Genetic Algorithm",
    description = "The objective of this " +
        "problem is to optimize the cost of construction of the network by " +
        "varying the diameter of the pipe in order to ensure a minimum level of pressure."
    @Parameters operators = {
        @OperatorInput(displayName = "Selection Operator", value = {
            @OperatorOption(displayName = "Uniform Selection", value = UniformSelection.class)
        }),
        @OperatorInput(displayName = "Crossover Operator", value = {
            @OperatorOption(displayName = "Integer Single Point Crossover", value = IntegerSinglePointCrossover.class),
            @OperatorOption(displayName = "Integer SBX Crossover", value = IntegerSBXCrossover.class)
        })
    }, //
    @OperatorInput(displayName = "Mutation Operator", value = {
        @OperatorOption(displayName = "Integer Simple Random Mutation", value = IntegerSimpleRandomMutation.class),
        @OperatorOption(displayName = "Integer Polynomial Mutation", value = IntegerPolynomialMutation.class),
        @OperatorOption(displayName = "Integer Range Random Mutation", value = IntegerRangeRandomMutation.class)
    }),
    files = {
        @FileInput(displayName = "Gamma *")
    },
    numbers = {
        @NumberInput(displayName = "Independent run", defaultValue = 5),
        @NumberInput(displayName = "Min pressure", defaultValue = 30),
        @NumberInput(displayName = "Population Size", defaultValue = 100)
    },
    numbersToggle = {
        @NumberToggleInput(groupID = "Finish Condition",
            displayName = "Max number of evaluation",
            defaultValue = 25000),
        @NumberToggleInput(groupID = "Finish Condition",
            displayName = "Number of iteration without improvement",
            defaultValue = 100)
    }
}
public PipeOptimizingRegister(Object selectionOperator, Object crossoverOperator, Object mutationOperator,
    File gama, int independentRun, int minPressure,
    int populationSize, int maxEvaluations, int numberWithoutImprovement
    throws Exception {
    // conversión de los object al tipo de operador específico.
}
}

```

Figura 4.16: Uso de anotaciones en la clase que hereda *Registrable* para construir la interfaz de configuración del problema

4.7. Pruebas

En esta ultima fase del proceso de desarrollo se procede a evaluar las funcionalidades implementadas.

Para la evaluación de las funcionalidades de la aplicación se aplican una serie de casos de prueba. Estos casos de prueba se dividen en dos categoría. Los casos de prueba utilizados para pruebas automatizadas y aquellos casos de prueba utilizados para pruebas manuales.

Los casos de prueba automatizados son aquellos realizados utilizando herramientas que permiten automatizar las tareas necesarias para evaluar los elementos que componen el software. Estas pruebas se llevan a cabo sobre los elementos del software como las clases o funciones implementadas. Estos casos de prueba se identificaron usando diversos métodos de testeo como herramienta para especificar los casos que deben ser evaluados. Los métodos utilizados principalmente para identificar los casos de prueba fueron los de caja negra y los de caja blanca.

Los caso de prueba manuales se refieren a aquellas pruebas realizadas con la ayuda humana. Es decir, realizados personalmente por un ser humano. Estos casos de prueba son utilizados principalmente para comprobar la funcionalidades de la interfaz gráfica donde automatizar las pruebas tiene una mayor complejidad.

A continuación se presentan las pruebas realizadas que abarcan a cada una de las funcionalidades escogidas.

Funcionalidad 1 : El Cuadro 4.8 muestra la especificación formal para el caso de prueba manual realizado sobre esta funcionalidad.

Funcionalidad 2 : Los Cuadros 4.9 y 4.10 muestran la especificación para los casos de prueba manuales de esta funcionalidad.

Funcionalidad 3 : Los Cuadros 4.11, 4.12, 4.13 y 4.14 muestran la especificación para los casos de prueba manuales de esta funcionalidad.

Cuadro 4.8: Especificación caso de prueba manual MT001

Test ID:	MT001
Título:	Visualización de la red.
Característica:	Mostrar visualización de la red.
Objetivo:	Confirmar que la red puede ser leída desde un archivo ".inp" y ser visualizada en la aplicación.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	inp: hanoi-Frankenstein.inp
Acciones de prueba:	1. Abrir JHawanetFramework 2. Cargar archivo de red.
Resultados esperados:	El sistema muestra la red leída desde un archivo inp gráficamente en la aplicación.

Cuadro 4.9: Especificación caso de prueba manual MT002

Test ID:	MT002
Título:	Optimización monoobjetivo realizada completamente.
Característica:	Realizar simulación monoobjetivo.
Objetivo:	Confirmar que se puede llevar a cabo la resolución del problema <i>PipeOptimizing</i> sobre la red abierta.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	independentRun = 10 Archivo inp: hanoi-Frankenstein.inp Archivo gama: hanoiHW.Gama
Acciones de prueba:	1. Abrir JHawanetFramework 2. Cargar archivo de red. 3. Seleccionar el problema <i>PipeOptimizing</i> del menú. 4. Configurar el problema usando la ventana de configuración. 5. Realizar la optimización.
Resultados esperados:	Al terminar la optimización el sistema muestra una interfaz con las soluciones generadas por la optimización. Debe haber tantas soluciones como el numero de configuraciones independientes (<i>independentRun</i>).

Cuadro 4.10: Especificación caso de prueba manual MT003

Test ID:	MT003
Título:	Optimización monoobjetivo cancelada.
Característica:	Realizar simulación monoobjetivo.
Objetivo:	Confirmar que se puede llevar a cabo la resolución del problema <i>PipeOptimizing</i> sobre la red abierta y que se puede cancelar el proceso cerrando la ventana o pulsando cancelar.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: hanoi-Frankenstein.inp Archivo gama: hanoiHW.Gama
Acciones de prueba:	1. Abrir JHawanetFramework 2. Cargar archivo de red. 3. Seleccionar el problema <i>PipeOptimizing</i> del menú. 4. Configurar el problema usando la ventana de configuración. 5. Realizar la optimización.
Resultados esperados:	Al cancelar la búsqueda de soluciones la ventana de estado indica que la optimización a sido detenida.

Cuadro 4.11: Especificación caso de prueba manual MT013

Test ID:	MT013
Título:	Simulación hidráulica sobre red de un solo tiempo.
Característica:	Realizar simulación con configuración de archivo de red.
Objetivo:	Confirmar que se puede realizar la simulación utilizando los valores del archivo de red.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: hanoi-Frankenstein.inp Archivo gama: hanoiHW.Gama
Acciones de prueba:	1. Abrir JHawanetFramework 2. Cargar archivo de red. 3. Pulsar botón <i>Execute</i> de la ventana principal.
Resultados esperados:	Ejecución realizada sin ningun error.

Cuadro 4.12: Especificación caso de prueba manual MT014

Test ID:	MT014
Titulo:	Ver resultados de la simulación hidráulica de un solo tiempo
Característica:	Realizar simulación con configuración de archivo de red.
Objetivo:	Confirmar que se puede visualizar los resultados de la simulación realizada.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: hanoi-Frankenstein.inp Archivo gama: hanoiHW.Gama
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework 2. Cargar archivo de red. 3. Pulsar botón <i>Execute</i> de la ventana principal. 4. Pulsar botón <i>Results</i> de la ventana principal.
Resultados esperados:	Una interfaz que permite seleccionar si se quieren ver los resultados para los enlaces o los nodos.

Cuadro 4.13: Especificación caso de prueba manual MT015

Test ID:	MT015
Titulo:	Simulación hidráulica sobre red de mas de un tiempo.
Característica:	Realizar simulación con configuración de archivo de red.
Objetivo:	Confirmar que se puede realizar la simulación utilizando los valores del archivo de red.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: Vanzyl.inp Archivo gama: VanzylConfiguration.json
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework 2. Cargar archivo de red. 3. Pulsar botón <i>Execute</i> de la ventana principal.
Resultados esperados:	Ejecución realizada sin ningun error.

Cuadro 4.14: Especificación caso de prueba manual MT016

Test ID:	MT016
Titulo:	Ver resultados de la simulación hidráulica de mas de un tiempo
Característica:	Realizar simulación con configuración de archivo de red.
Objetivo:	Confirmar que se puede visualizar los resultados de la simulación realizada.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: Vanzyl.inp Archivo gama: VanzylConfiguration.json
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework 2. Cargar archivo de red. 3. Pulsar botón <i>Execute</i> de la ventana principal. 4. Pulsar botón <i>Results</i> de la ventana principal.
Resultados esperados:	Una interfaz que permite seleccionar si se quieren ver los resultados para los enlaces o los nodos. Adicionalmente, permite escoger el tiempo de simulación y listar los resultados para todos los tiempos de un elemento de la red específico.

La especificación de las restantes pruebas realizadas se encuentra en el **Anexo D**.

5. Evaluación de la solución

En este capítulo se da a conocer la metodología de evaluación y su aplicación, detallando las fases y los resultados de éstas, con el fin de evaluar la solución implementada.

5.1. Metodología de evaluación

La metodología que se ha escogido para realizar este análisis consiste en el estudio de caso. Para llevar a cabo este estudio de caso se comienza realizando el diseño del caso de estudio, luego se realiza la recolección de datos, el análisis de los datos recolectados y finalmente de reportan los resultados junto con las conclusiones del estudio.

En el diseño del caso, se define el caso a estudiar, los objetivos, el protocolo para conducir el estudio, las características que se quieren evaluar y los sujetos de prueba o unidad de análisis.

En la recolección de datos, se aplica el instrumento al personal encuestado.

En la etapa de análisis, se procesan los datos presentando una serie de gráficos que ayudan a visualizar de mejor manera la información recolectada.

Finalmente, durante la etapa de reporte se presentan las conclusiones de la aplicación del caso de estudio.

5.2. Diseño del estudio de caso

En esta sección, como se mencionó anteriormente, se detalla el plan de acción previo a la aplicación del caso.

5.2.1. Elección del caso

El caso a evaluar consiste en el software desarrollado durante este proyecto de titulación. Como ya se sabe, es un software que permite buscar soluciones a problemas hidráulicos utilizando algoritmos metaheurísticos.

5.2.2. Objetivos de la investigación

El estudio que se quiere llevar a cabo es un estudio del tipo cualitativo, transversal y descriptivo, con el cual se busca comprender como el sistema desarrollado se desempeña en la práctica y su grado de aceptación.

Como pregunta de investigación se establece la siguiente:

“¿Cómo el sistema desarrollado trabaja en la práctica?”

5.2.3. Características a evaluar

Las características que se buscan evaluar mediante la aplicación del caso de estudio son las siguientes:

- **Funcionalidad:** Con esta característica se busca evaluar lo que la aplicación puede hacer, es decir, si el sistema cumple con las operaciones que se busca realizar con el programa.
- **Facilidad de uso:** Esta característica busca medir la usabilidad del software, es decir, que tan fácil es de usar la aplicación.
- **Utilidad:** Con este criterio se busca medir el valor de la aplicación. Con valor, nos referimos a si las funcionalidades implementadas por el sistema son de utilidad para quien haga uso de solución.
- **Utilidad del manual de usuario:** Con este criterio se busca evaluar el manual de usuario realizado y si este fue de ayuda para aprender y poder conocer la aplicación.

La medición de cada una de estas características se lleva a cabo usando encuestas.

5.2.4. Protocolo para conducir el estudio de caso

El objetivo de este protocolo es establecer una guía para realizar la recolección de datos.

Debido a la complejidad de coordinar y agendar una reunión se optó por realizar la recolección de datos de manera asíncrona.

El protocolo para llevar a cabo la evaluación del sistema consiste en:

1. Enviar la aplicación junto con su manual a los participantes del estudio.
2. Dar un tiempo para que el participante del estudio se interiorice con la aplicación.
3. Enviar la encuesta a los participantes.

El instrumento utilizado para evaluar la solución se encuentra en el **Anexo E**.

5.2.5. Unidad de análisis

La recopilación de datos se lleva a cabo sobre profesionales con conocimientos tanto en el área computacional, hidráulica y metaheurísticas.

5.3. Consideraciones preliminares

Consideraciones técnicas: En esta sección se mencionan algunas consideraciones preliminares técnicas como de usuario.

- Para utilizar el software se requiere el sistema operativo Window con la versión de Java 1.8.
- Para crear una nueva red se debe usar un programa externo. Este programa es Epanet y la red debe ser exportada a un archivo .inp. Debido a que la versión en español y la en inglés de Epanet crean el archivo .inp cambiando algunas palabras claves que forman parte del archivo de configuración de red exportado, se optó por utilizar el formato de la versión en inglés de Epanet dentro del programa desarrollado.

- Para visualizar los resultados de la exportación a un documento de Excel, se requiere dicho programa o alguna alternativa que permita visualizar las hojas de cálculo con extensión .xlsx.

Consideraciones para los usuarios:

- Para poder implementar nuevos problemas y algoritmos se requiere tener conocimientos en hidráulica y programación, así como también en metaheurísticas. Si no se requiere implementar nuevos elementos, los conocimientos en programación no son tan necesarios, mientras que los de metaheurísticas e hidráulica sí, con el objetivo de saber como interpretar los resultados.

5.4. Recolección de datos

En esta sección se presentan los participantes del estudio y se detallan las características del instrumento utilizado para la obtención de datos.

Participantes

Los participantes del proceso de recolección de datos fueron los siguientes:

Evaluador: Gabriel Sanhueza

Usuarios:

Todos los usuarios participan en el proyecto de investigación Fondecyt *Optimization of real-world water distribution systems and hydraulic elements using computational fluid dynamics (cfd) and evolutionar y algorithms*.

- Yamisleydi Salgueiro: Doctorado en Ciencias Técnicas (Inteligencia Artificial) y Co-investigadora del proyecto.
- Marco Alsina: Doctorado en Ciencias de la Ingeniería; especialista hidráulico; profesor Ingeniería Civil en Obras Civiles y Co-investigador del proyecto.
- Sergio Silva: Ingeniero Civil en Computación, personal técnico del proyecto.

Instrumentos para la recolección de los datos

Para el proceso de recolección de los datos se utilizaran encuestas. La encuesta esta diseñada para evaluar las características mencionadas en la subsección 5.2.3. La funcionalidad y la utilidad se evalúan usando la escala de Likert. En cuanto a la usabilidad, este se evalúa usando la escala de usabilidad de SUS.

La escala de usabilidad de SUS cuenta con 10 preguntas, las cuales a efecto de la presente evaluación son:

1. Creo que usaría esta aplicación frecuentemente.
2. Encuentro esta aplicación innecesariamente complejo.
3. Creo que la aplicación fue fácil de usar.
4. Creo que necesitaría ayuda de una persona con conocimientos técnicos para usar esta aplicación.
5. Las funciones de esta aplicación están bien integradas.
6. Creo que la aplicación es muy inconsistente.
7. Imagino que la mayoría de la gente aprendería a usar esta aplicación en forma muy rápida.
8. Encuentro que la aplicación es muy difícil de usar.
9. Me siento confiado al usar esta aplicación.
10. Necesité aprender muchas cosas antes de ser capaz de usar esta aplicación.

La encuesta aplicada se encuentra en el **Anexo E**.

5.5. Análisis de datos

En esta sección se presenta el análisis de los datos. El análisis se realiza de manera independiente por cada una de las características a evaluar, en donde se muestran los resultados utilizando herramientas como gráficos y tablas. Adicionalmente, se provee una interpretación de los resultados.

A continuación se presentan los resultados por cada categoría evaluada.

5.5.1. Funcionalidad

En la Figura 5.1 se presentan los resultados de la evaluación de la funcionalidad en forma de un gráfico de barras. Mientras, que en el Cuadro 5.1 se muestra la tabla con el porcentaje obtenido por cada alternativa de la evaluación.

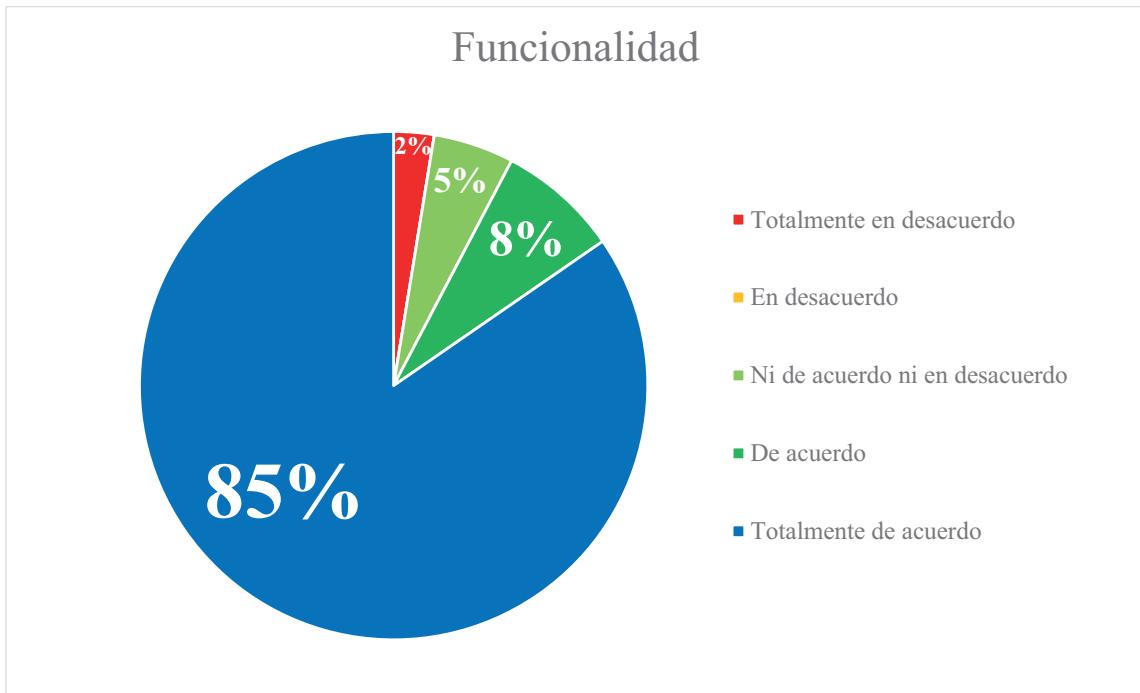


Figura 5.1: Gráfico circular de los resultados de la evaluación de funcionalidad de la aplicación

Cuadro 5.1: Resultados de la evaluación de la funcionalidad

Respuesta	Porcentaje Obtenido
Totalmente en desacuerdo	2.6 %
En desacuerdo	0.0 %
Ni de acuerdo ni en desacuerdo	5.1 %
De acuerdo	7.7 %
Totalmente de acuerdo	84.6 %
Total	100 %

Como se puede ver en la Figura 5.1 y en el Cuadro 5.1, el 93 % de las respuestas están “de acuerdo” y “totalmente de acuerdo” con el hecho de que las funcionalidades que incorpora la aplicación han sido implementadas correctamente. Por otro lado, el 5 % de las respuestas apuntan a que no se está “ni de acuerdo ni en desacuerdo” con el hecho de si las funcionalidades están correctamente integradas. Mientras que el 2 % de las respuestas apuntan a que las funcionalidades no fueron integradas correctamente o que había una carencia de alguna de ellas.

5.5.2. Usabilidad

El Cuadro 5.2 presenta la escala de clasificación utilizada para comparar el puntaje SUS.

Cuadro 5.2: Escala de rangos de *SUS Score* [39]

<i>SUS Score</i>	Calificación
> 80,3	Excelente
68 - 80,3	Buena
68	Regular
51 - 68	Mala
< 51	Terrible

La aplicación del instrumento de medición dio como resultado un promedio:

$$\bar{x}_{SUS} = 79,17$$

con una desviación estándar de :

$$s_{SUS} = 9,46$$

De esto se puede concluir que en el aspecto de usabilidad la aplicación se le califica como ”Buena” .

5.5.3. Utilidad

La Figura 5.2 y el Cuadro 5.3 muestran los resultados obtenidos de la evaluación de la utilidad de la aplicación.

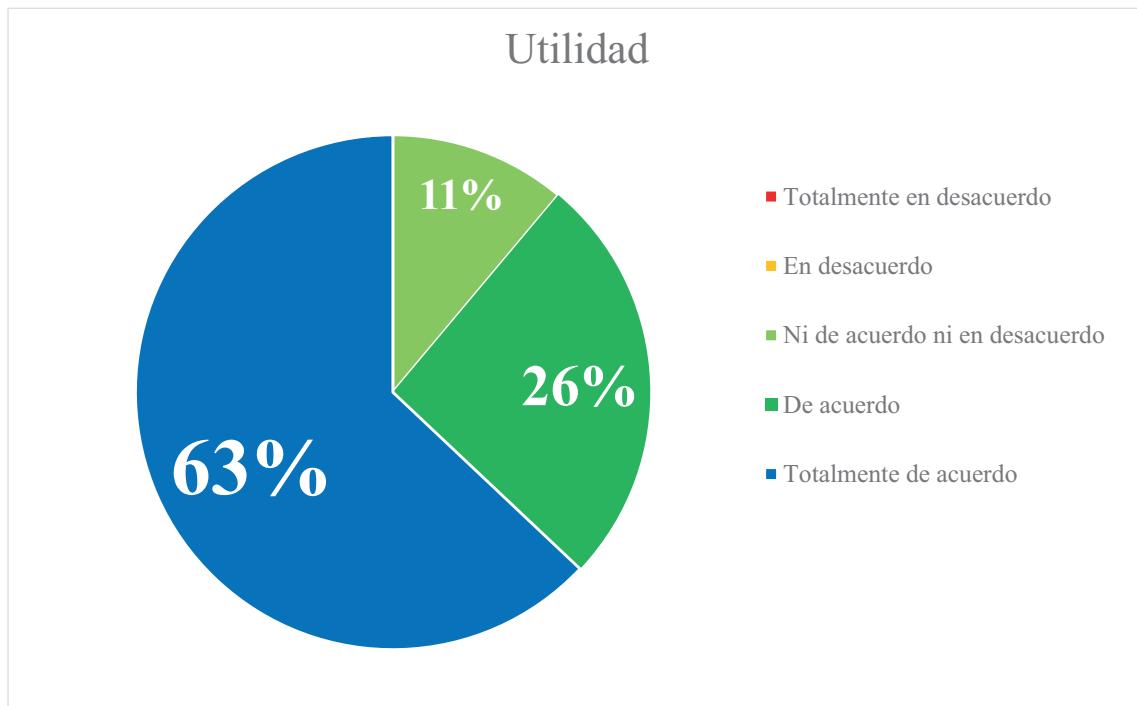


Figura 5.2: Gráfico circular de los resultados de la evaluación de utilidad de la aplicación

Cuadro 5.3: Resultados de la evaluación de la utilidad

Respuesta	Porcentaje Obtenido
Totalmente en desacuerdo	0.0 %
En desacuerdo	0.0 %
Ni de acuerdo ni en desacuerdo	11.1 %
De acuerdo	25.9 %
Totalmente de acuerdo	63.0 %
Total	100 %

De la Figura 5.2 y Cuadro 5.3 se observa que el 89 % de las respuestas indican que se está “totalmente de acuerdo” o “de acuerdo” con que las funcionalidades implementadas en la aplicación son de utilidad para los encuestados. Mientras que un 11 % indican que no se está “ni se acuerda ni en desacuerdo” con si realmente

son útiles o no.

5.5.4. Utilidad del manual de usuario

Los resultados de la evaluación de la utilidad del manual de usuario se presentan en la Figura 5.3 y el Cuadro 5.4.

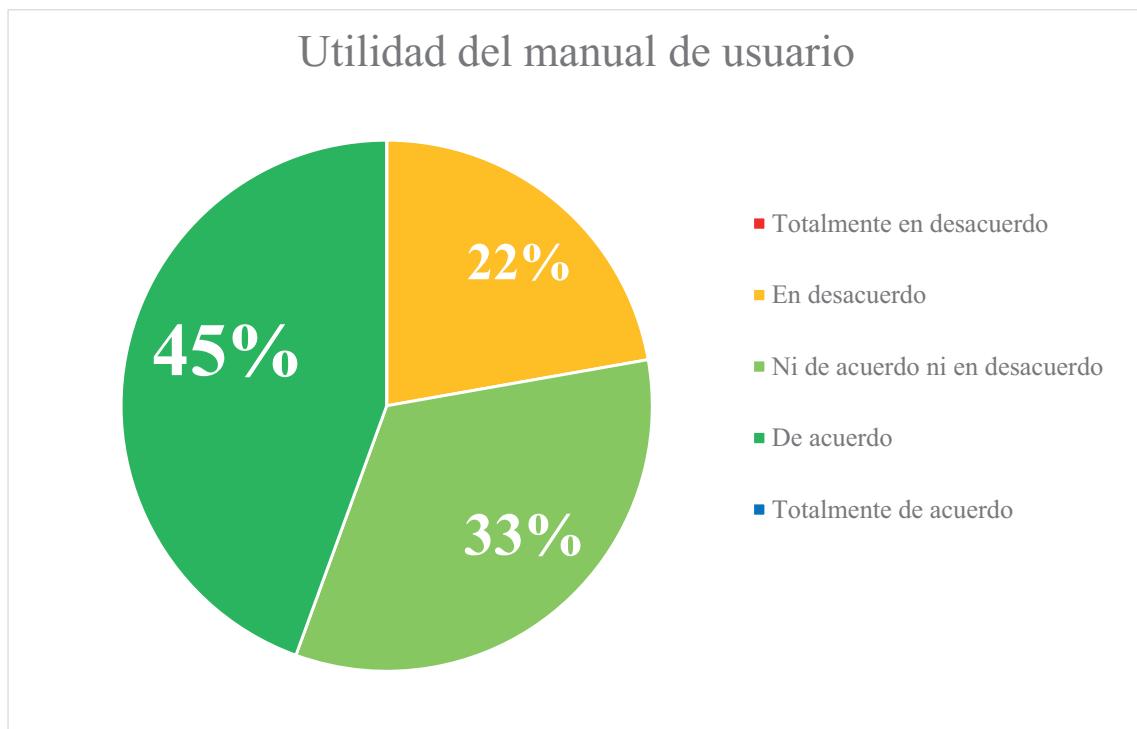


Figura 5.3: Gráfico circular de los resultados de la evaluación de la utilidad del manual de usuario

Cuadro 5.4: Resultados de la evaluación de la utilidad del manual de usuario

Respuesta	Porcentaje Obtenido
Totalmente en desacuerdo	0.0 %
En desacuerdo	22.2 %
Ni de acuerdo ni en desacuerdo	33.3 %
De acuerdo	44.4 %
Totalmente de acuerdo	0.0 %
Total	100 %

Con respecto a la utilidad del manual de usuario, de acuerdo a los resultados presentados en la Figura 5.3 y el Cuadro 5.4, se puede apreciar que el 45 % de las respuestas apuntan a que se está “de acuerdo” con que el manual de usuario presentado junto con el software fue de utilidad para comprender como utilizarlo y realizar ciertas tareas con éste. Por otro lado, un 33 % indica que no se está “ni de acuerdo ni en desacuerdo” con si el manual de usuario fue útil o no, mientras que el 22 % indica que se está en “desacuerdo” con la utilidad del manual y por lo tanto que es deficiente y debiera ser mejorado.

Las encuestas respondida por cada usuario se encuentran en el **Anexo ??**.

5.6. Conclusiones del estudio

Durante esta sección se presentan las conclusiones obtenidas a partir de la aplicación del caso de estudio.

A partir de los resultados obtenidos se puede concluir que la aplicación cumple con los criterios de funcionalidad, usabilidad y utilidad. Por lo tanto, puede ser ocupada en un contexto real y ser ocupada para buscar soluciones a problemas de RDA.

Por otra parte, si bien el manual de usuario sirvió para utilizar y comprender la aplicación este debe ser mejorado incorporando más detalles y ejemplos de las funcionalidades explicadas.

6. Conclusiones Y Trabajos Futuros

En este capítulo se presentan las conclusiones del proyecto así como las propuestas de trabajo futuro que pueden ser realizados sobre la herramienta.

6.1. Conclusiones

Se ha desarrollado un software que permite optimizar los procesos de diseño y operación en redes de agua potable. Dicho software, incorpora por defecto dos problemas y dos algoritmos. Los problemas implementados son el de diseño de RDA basado en la selección del diámetro de tuberías y el problema del régimen de bombeo. En cuanto a los algoritmos añadidos, se encuentran el Algoritmo Genético y el algoritmo NSGAII.

Los objetivos definidos para este trabajo son exitosamente logrados en el tiempo planificado.

La aplicación de la metodología se llevo a cabo realizando las cuatro fases correspondientes a requisitos, diseño, implementación y pruebas en cada una de las iteraciones. Al finalizar el periodo de desarrollo todas las funcionalidades solicitadas se encuentran implementadas.

La metodología de desarrollo utilizada fue la adecuada dando como resultado documentación extra que puede ser consultada por aquellos que deseen continuar mejorando el sistema en futuros trabajos. Los documentos generados son la especificación de requisitos (Anexo A), la especificación del diseño (Anexo B); el manual de usuario (Anexo C); y la especificación de los casos de prueba (Anexo D).

Gracias a la utilización de los patrones de Inversión de Control y Inyección de Dependencias, así como la utilización de las funcionalidades de Java, *Java Reflection* y *Java Annotation*, se logra mejorar la capacidad de extensibilidad del programa facilitando la incorporación de nuevos algoritmos y problemas.

En relación a la evaluación de la solución se concluye que la aplicación incorpora las funcionalidades solicitadas en un principio por los interesados, el software es evaluado como “bueno” en cuanto a la usabilidad de acuerdo a la escala SUS utilizada y es de utilidad para los usuarios que se desempeñan en el área de redes de agua potable. Sin embargo, el manual de usuario debe ser mejorado incorporando más detalles de la aplicación.

La aplicación de encuentra disponible en el repositorio de GitHub <https://github.com/EinherjarSt/ProyectoDeMemoria>.

6.2. Trabajo futuro

Una vez terminado el desarrollo del proyecto se identificó por parte del desarrollador, los colaboradores y los sujetos del estudio realizado, una serie de cadencias y detalles que pueden ser mejorados, así como algunas funcionalidades extras que incrementan el valor de la aplicación. Éstas se listan a continuación:

- Agregar nuevos algoritmos, operadores y problemas al sistema.
- Recortar el número de decimales con que se presentan los resultados en la aplicación.
- Exportar el gráfico de la red como una imagen vectorial.
- Exportar el gráfico de soluciones para uno y dos objetivos como una imagen vectorial.
- Cambiar el tamaño de los iconos de la red cuando se posicione el cursor sobre éstos.
- Permitir agregar fórmulas utilizando Latex en la descripción de los algoritmos.
- Permitir utilizar distintos algoritmos en un mismo experimento.
- Incorporar métricas de comparación entre algoritmos.

- Mostrar en la interfaz los patrones de demanda y bombeo de la red cuanto ésta los especifique.
- Permitir restablecer los valores por defecto en la pestaña de configuración del problema.
- Incorporar una cola de trabajo para ejecutar más de una optimización a la vez.

Bibliografía

- [1] H. A. Basha and B. G. Kassab. Analysis of water distribution systems using a perturbation method. *Applied Mathematical Modelling*, 20(4):290–297, 1996.
- [2] Gabriel Pereyra, Daniel Pandolfi, and Andrea Villagra. Diseño y optimización de redes de distribución de agua utilizando algoritmos genéticos. *Informes Científicos Técnicos - UNPA*, 9(1):37–63, 2017.
- [3] Jimmy H. Gutiérrez-Bahamondes, Yamisleydi Salgueiro, Sergio A. Silva-Rubio, Marco A. Alsina, Daniel Mora-Meliá, and Vicente S. Fuertes-Miquel. jHawanet: An open-source project for the implementation and assessment of multi-objective evolutionary algorithms on water distribution networks. *Water (Switzerland)*, 11(10), 2019.
- [4] Juan J. Durillo, Antonio J. Nebro, and Enrique Alba. The jMetal framework for multi-objective optimization: Design and architecture. *2010 IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*, pages 1–8, 2010.
- [5] Jimmy H. Gutiérrez-Bahamondes, Daniel Mora-Melia, Yamisleydi Salgueiro, Sergio A. Silva-Rubio, and Pedro L. Iglesias-Rey. jMetal como herramienta de comparación de algoritmos multiobjetivo aplicados a la ingeniería hidráulica. Number November, 2019.
- [6] Pedro L. Iglesias Rey, F. Javier Martínez-Solano, Vicente S. Fuertes-Miquel, and Rafael Pérez-García. Algoritmo genético modificado para diseño de redes de abastecimiento de agua. *IV Seminário Hispano-Brasileiro sobre Sistemas de Abastecimento Urbano de Água*, (May 2014):1–16, 2004.

- [7] Roger S Pressman. *Software Engineering A Practitioner's Approach*. 7 edition, 2009.
- [8] Adel Alshamrani and Abdullah Bahattab. A comparison between three SDLC models waterfall model, spiral model, and incremental/iterative model. *IJCSI International Journal of Computer Science Issues*, 12(1):106–111, 2015.
- [9] R Victor. Iterative and incremental development: A brief history. 2003.
- [10] Susan M. Mitchell and Carolyn B. Seaman. A comparison of software cost, duration, and quality for waterfall vs. iterative and incremental development: A systematic review. *2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009*, (February 2008):511–515, 2009.
- [11] Robert C Martin. Iterative and incremental development (IID). *Design*, (Iid), 1999.
- [12] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.
- [13] C L Sabharwal. Java, Java, Java. *IEEE Potentials*, 17(3):33–37, 1998.
- [14] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. The Java® Language Specification Java SE 8 Edition, 2015.
- [15] Mathias Braux and Jacques Noyé. Towards partially evaluating reflection in Java. *ACM SIGPLAN Notices*, 34(11):2–11, 1999.
- [16] Henrique Rocha and Marco Tulio Valente. How annotations are used in Java: An empirical study. *SEKE 2011 - Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering*, (June 2014):426–431, 2011.
- [17] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys (CSUR)*, 17(4):471–523, 1985.
- [18] Modeling T H E State and Laser Scanning Technology. Usage of Dependency Injection within different frameworks. (March), 2020.
- [19] Shrinidhi R. Hudli and Raghu V. Hudli. A Verification Strategy for Dependency Injection. *Lecture Notes on Software Engineering*, (January 2013):71–74, 2013.

- [20] Lewis Rossman. *EPANET 2.0 en español. Análisis hidráulico y de calidad del agua en redes de distribución de agua. manual del usuario.* 2017.
- [21] Lewis A. Rossman. The EPANET Programmer's Toolkit for Analysis of Water Distribution Systems. In *WRPMD'99*, pages 1–10, Reston, VA, jun 1999. American Society of Civil Engineers.
- [22] VS Fuertes, J García-Serra, PL Iglesias, G López, FJ Martínez, and R Pérez. Modelación y diseño de redes de abastecimiento de agua. *Fluid Mechanics Group, Polytechnic University of Valencia, Spain*, 2002.
- [23] Daniel Mora. Diseño de redes de distribución de agua mediante algoritmos evolutivos. análisis de eficiencia. 2012.
- [24] Xin She Yang. Metaheuristic optimization. *Scholarpedia*, 6(2011):11472, 2015.
- [25] Omid Bozorg-Haddad, Mohammad Solgi, and Hugo A Loaiciga. *Meta-heuristic and evolutionary algorithms for engineering optimization*. John Wiley & Sons, Incorporated, Newark, United States, 2017.
- [26] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.
- [27] Zahra Beheshti and Siti Mariyam Shamsuddin. A Review of Population-based Meta-Heuristic Algorithm. (March 2013), 2015.
- [28] David a. Van Veldhuizen and Gary B Lamont. Evolutionary Computation and Convergence to a Pareto Front. *Late Breaking Papers at the Genetic Programming 1998 Conference*, pages 221–228, 1998.
- [29] Marc H.J. Romanycia and Francis Jeffry Pelletier. What is a heuristic? *Computational Intelligence*, 1(1):47–58, 1985.
- [30] Sean Luke. *Essentials of metaheuristics*. Lulu, second edition, 2013.
- [31] Dorothea Heiss-Czedik. An introduction to genetic algorithms. *Artificial Life*, 3(1):63–65, 1997.

- [32] Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
- [33] Adam Slowik and Halina Kwasnicka. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, (March), 2020.
- [34] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 2002.
- [35] Yasaman Makaremi, Ali Haghghi, and Hamid Reza Ghafouri. Optimization of Pump Scheduling Program in Water Supply Systems Using a Self-Adaptive NSGA-II; a Review of Theory to Real Application. *Water Resources Management*, 31(4):1283–1304, 2017.
- [36] Pino V. Edwin, Valle C. Angely, Condori P. Franz, Mejia M. Jesus, Chavarri V. Eduardo, and Alfaro R. Luis. Diseño óptimo de redes de distribución de agua usando un software basado en microalgoritmos genéticos multiobjetivos. *Ribagua*, 4(1):6–23, 2017.
- [37] Darshan Mehta, Vipin Yadav, Keyur Prajapati, and Sahita Waikhom. Design of optimal water distribution systems using WaterGEMS: A case study of Surat city. *E-proceedings of the 37th IAHR World Congress*, (December):1–8, 2017.
- [38] Donald Bell. UML basics : The component diagram. *IBM developerWorks*, (December):1–10, 2004.
- [39] Will T. Measuring and interpreting system usability scale (sus). <https://uiuxtrend.com/measuring-system-usability-scale-sus/>. Accedido 20-07-2020.

ANEXOS

A. Documento de especificación de requisitos



UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Documento de especificación de requisitos

**HERRAMIENTA PARA LA OPTIMIZACIÓN DE REDES DE
DISTRIBUCIÓN DE AGUA POTABLE**

TABLA DE CONTENIDOS

página

Tabla de Contenidos

1. Introducción	1
1.1. Propósito del Sistema	1
1.2. Alcance del proyecto	1
1.3. Contexto	2
1.4. Definiciones, Acrónimos y Abreviaturas	3
1.5. Referencias	3
2. Descripción General	5
2.1. Características de los Usuarios	5
2.2. Ambiente operacional de la solución	5
2.3. Relación con otros proyectos	6
2.4. Restricciones Generales	6
3. Requisitos	7
3.1. Requisitos de usuario	7
3.2. Requisitos de sistema	26
3.3. Matriz de Trazado Requisitos de Usuario vs. Requisitos de Software .	59

1. Introducción

El proyecto a desarrollar consiste en la creación de una herramienta que hace uso de algoritmos metaheurísticos para optimizar problemas existentes en redes de distribución de agua potable.

Este proyecto solo abarca dos problemas existentes dentro de las redes de distribución de agua potable.

Para el desarrollo de este proyecto se usan librerías ya existentes con el fin de reducir el tiempo de desarrollo. Estas librerías son epanet2.dll, desarrollada en C, y JNA, librería en Java para realizar llamadas a funciones nativas.

1.1. Propósito del Sistema

El proyecto consiste en el desarrollo de un sistema que permite simular y buscar soluciones a problemas presentes en las redes de distribución de agua potable haciendo uso de algoritmos metaheurísticos. Adicionalmente, el sistema es diseñado de tal forma que pueda ser extendido añadiendo nuevos problemas, algoritmos u operadores.

1.2. Alcance del proyecto

Al final del período de desarrollo la herramienta debe contar con las siguientes prestaciones.

- El sistema permite la carga y la visualización de la red gráficamente.
- El sistema implementa por defecto dos algoritmos. Uno de ellos monoobjetivo y el otro multiobjetivo. El algoritmo monoobjetivo corresponde al Algoritmo

Genético (GA) mientras que el multiobjetivo a *Non-Dominated Sorting Genetic Algorithm II* (NSGAII).

- El sistema proporciona dos problemas ya implementados uno monoobjetivo y el otro multiobjetivo. El Algoritmo Genético se utiliza para generar soluciones para el problema monoobjetivo con el fin de optimizar el costo de inversión de las tuberías. Por otro lado, el algoritmo NSGAII aborda el problema multiobjetivo con el objetivo de optimizar los costos energéticos y los costos de mantenimiento de los equipos de bombeo.
- El sistema permite visualizar y guardar las soluciones de los algoritmos en un archivo.
- El sistema permite que el usuario agregue nuevos algoritmos, operadores o problemas sin tener que modificar la interfaz de usuario.

Este proyecto no contempla la creación de la red por lo que estas deberán ser ingresadas como entradas al programa. La creación de las redes puede realizarse a través de EPANET. Además, esta herramienta únicamente podrá ser ocupada en equipos cuyo sistema operativo sea Windows de 64bits debido a que se realizan llamadas a librerías nativas las cuales no son multiplataforma. Adicionalmente, el equipo debe contar con Java 1.8.

1.3. Contexto

Este sistema es desarrollado utilizando el lenguaje de programación Java. Debido a que este es un lenguaje ampliamente utilizado y que cuenta con un gran soporte y una gran comunidad que lo utiliza.

Como motor de cálculo para llevar a cabo las simulaciones se utiliza una librería desarrollada en C, que cuenta con funciones para llevar a cabo simulaciones de redes de agua potable. El nombre de esta librería es epanet2.dll. Las funciones que incorpora esta librería se encuentran explicadas en [4].

Desde Java se realizan las llamadas a librerías nativas usando la librería JNA. Esta librería cuenta con las clases y métodos necesarios para poder acoplar este sistema a la librería epanet2.dll desarrollada en C y que es usada como motor de cálculo para llevar a cabo las simulaciones. Puesto que una de las funcionalidades del

sistema es permitir la ejecución de algoritmos metaheurísticos, se toma como base la arquitectura presentada por el framework JMetal.

JMetal es un para la optimización multiobjetivo con metaheurísticas. Su arquitectura inicial [1] involucraban una serie de problemas y dificultaban la realización de ciertas acciones que eran recurrentes. Además, esta no hacia uso de las novedades incorporadas por Java como los genéricos. Es por esto, que posteriormente fue rediseñada, haciendo uso de patrones de diseño, principios de la programación orientada a objetos y aprovechando las características del lenguaje Java. Este rediseño se presenta en [3] y es el que se toma como base para la utilización de algoritmos metaheurísticos durante este proyecto.

El contexto en el que se desenvuelve este sistema es en ambientes universitario, de investigación y en el ambiente laboral.

1.4. Definiciones, Acrónimos y Abreviaturas

GA: Algoritmo Genético

NSGAI: Non-dominated Sorting Genetic Algorithm

JNA: Java Native Access

1.5. Referencias

- [1] Juan J. Durillo, Antonio J. Nebro, and Enrique Alba. The jMetal framework for multi-objective optimization: Design and architecture. *2010 IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*, pages 1–8, 2010.
- [2] Pedro L. Iglesias Rey, F. Javier Martínez-Solano, Vicente S. Fuertes-Miquel, and Rafael Pérez-García. Algoritmo genético modificado para diseño de redes de abastecimiento de agua. *IV Seminário Hispano-Brasileiro sobre Sistemas de Abastecimento Urbano de Água*, (May 2014):1–16, 2004.
- [3] Antonio J. Nebro, Juan J. Durillo, and Matthieu Vergne. Redesigning the jMetal multi-objective optimization framework. In *GECCO 2015 - Companion Publication of the 2015 Genetic and Evolutionary Computation Conference*, 2015.

- [4] Lewis Rossman. *EPANET 2.0 en español. Análisis hidráulico y de calidad del agua en redes de distribución de agua. manual del usuario.* 2017.

2. Descripción General

En esta sección se describe las características de los usuarios que harán uso del sistema. Además, se menciona el ambiente operacional de la solución y la relación que este proyecto tiene con otros proyectos. Finalmente, también se menciona las restricciones generales de la herramienta a desarrollar.

2.1. Características de los Usuarios

Este sistema solo cuenta con un tipo de usuario el cual tendrá acceso a todas las funcionalidades. Se espera que los usuarios que utilicen este sistema sean ingenieros, investigadores u personas que cuenten con el conocimiento básico acerca de redes de distribución de agua potable y metaheurísticas, que les servirá para interpretar los resultados generados por los algoritmos.

2.2. Ambiente operacional de la solución

El ambiente operacional en el que se desarrolla el sistema es el siguiente:

- Intel(R) Core(TM) i7-7700HQ CPU @ 2.80Ghz 2.8Ghz
- RAM 16GB DDR4
- HDD 7200rpm 1T
- SSD 256GB PCIe NVME M.2
- Windows 10 x64
- NVIDIA GeForce GTX 1050

2.3. Relación con otros proyectos

El sistema depende de la librería nativa epanet2_64bit.dll, ya que usa esta librería como motor de cálculo. La librería cuenta con 54 funciones dentro de las cuales se encuentran funciones para correr las simulaciones, modificar y obtener las configuraciones de la red, modificar los elementos que conforman la red y generar reportes.

Las llamadas desde Java a la librería nativa serán realizadas a través de la librería *Java Native Access* (JNA).

Adicionalmente, el sistema toma la arquitectura utilizada por el framework de optimización multiobjetivo Jmetal como base para agregar los algoritmos, operadores y problemas. Esta arquitectura será modificada según se necesite para satisfacer los requisitos del sistema.

2.4. Restricciones Generales

- La red debe ser ingresada como entrada al programa a través del archivo de configuración de red, cuya extensión debe ser inp.
- La herramienta solo está disponible para el sistema operativo Window de 64bits.
- El sistema debe ser ejecutado con Java 1.8 de 64bits.

3. Requisitos

En esta capítulo se presentan los requisitos capturados, los cuales están sujetos a cambios a medida que se avanza en las iteraciones.

3.1. Requisitos de usuario

A continuación, se presentarán los requisitos de usuarios que han sido obtenidos para el desarrollo de este proyecto

Durante la presentación de estos requisitos se hace referencia al archivo de configuración de red. Éste debe ser generado utilizando la aplicación Epanet y guardado con la extensión “inp” a partir de ahora nos referiremos al archivo de configuración de red simplemente como inp o archivo inp.

RU001 – Cargar una red.	
Descripción:	La red que es representada por el archivo .inp debe ser cargada en el programa para poder llevar a cabo la simulación.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	1
Tipo:	Funcional

RU002 – Resolver el problema monoobjetivo (<i>Pipe Optimizing</i>) usando un Algoritmo Genético.	
Descripción:	El Algoritmo Genético debe ser aplicado para resolver el problema monoobjetivo que tiene como función objetivo el costo de inversión y como variable de decisión el diámetro de las tuberías.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RU003 – Resolver el problema multiobjetivo (<i>Pumping Schedule</i>) usando un Algoritmo NSGAII.	
Descripción:	El algoritmo NSGAII debe ser aplicado al problema multiobjetivo cuyas funciones a optimizar son los costos energéticos y los costos de mantenimiento de los equipos de bombe (<i>Pumping Schedule</i>).
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	4
Tipo:	Funcional

RU004 – Visualizar red en una interfaz gráfica.	
Descripción:	Se debe mostrar en la interfaz gráfica una representación de la red (Un dibujo, etc) generada a partir de la información contenida en el archivo inp.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RU005 – Exportar los resultados de los algoritmos aplicados en dos archivos, uno para las variables y otro para los objetivos.	
Descripción:	Se deben poder exportar las soluciones generadas por la ejecución de los algoritmos sobre un problema a un conjunto de archivos. Específicamente, serían 2 archivos. El primer archivo debe guardar las variables de las soluciones mientras que el segundo archivo los objetivos de las soluciones.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RU006 – Implementar el operador IntegerSBXCrossover.	
Descripción:	El operador IntegerSBXCrossover es uno de los operadores de cruzamiento. Este operador en base a cálculos probabilísticos combina dos soluciones para crear dos nuevas soluciones hijas.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RU007 – Implementar el operador IntegerSinglePointCrossover.	
Descripción:	El operador IntegerSinglePointCrossover es un operador de cruzamiento. Viendo la solución como un vector, este operador toma dos soluciones y elige un punto a partir del cual los valores de una solución se intercambiarán con los valores de la otra solución. Este operador usa una probabilidad de cruzamiento y solamente realiza el intercambio de los valores en la solución cuando un número generado aleatoriamente es menor que la probabilidad de cruzamiento.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RU008 – Implementar el operador IntegerPolynomialMutation.	
Descripción:	El operador IntegerPolynomialMutation es un operador de mutación. Este operador de mutación usa cálculos probabilísticos para mutar algunos variables de decisión que forman parte de la solución.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RU009 – Implementar el operador IntegerSimpleRandomMutation.	
Descripción:	El operador IntegerSimpleRandomMutation es un operador de mutación. Este operador muta una variable de decisión cuando un número generado aleatoriamente es menor que la probabilidad de mutación establecida. El operador recorre cada variable de decisión realizando lo descrito anteriormente. La mutación mencionada por este operador consiste en cambiar el valor de la variable de decisión por otro valor aleatorio.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RU010 – Implementar el operador IntegerRangeRandomMutation.	
Descripción:	El operador IntegerRangeRandomMutation es un operador de mutación. Este operador muta una variable de decisión cuando un número generado aleatoriamente es menor que la probabilidad de mutación establecida. El operador recorre cada variable de decisión realizando lo descrito anteriormente. La mutación realizada por este operador consiste en cambiar el valor de la variable de decisión por otro valor aleatorio que se encuentre entre un rango establecido. Ejemplo: Variable de decisión: 3 Rango: 2 La variable de decisión después de aplicado el operador puede tomar un valor entre [1, 5].
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RU011 – Implementar el operador UniformSelection.	
Descripción:	<p>El operador UniformSelection [2] es un operador de selección. Este operador de selección ordena la población y asigna una probabilidad máxima y mínima a la mejor y peor solución, respectivamente. A las soluciones que se encuentran entre la mejor y la peor solución se le asigna una probabilidad entre el máximo y mínimo obtenido anteriormente. Si la probabilidad de la solución es mayor a 1.5 entonces la solución se duplica en la nueva población. Si la probabilidad está entre 0.5 y 1.5, entonces en la nueva población se agrega la solución solo una vez. Las soluciones cuya probabilidad es menor que 0.5 no aparecen en la nueva población.</p> <p>La probabilidad máxima puede ser calculada como $p_{max} = \beta/N_c$ mientras que la probabilidad mínima se calcula de acuerdo a $p_{min} = (2-\beta)/N_c$. Donde β es un número entre 1.5 y 2, y N_c es el número de soluciones presentes en el conjunto sobre el que se realizará la selección.</p> <p>La probabilidad de las soluciones intermedias se calcula de acuerdo a</p> $p_i = p_{min} + (p_{max} - p_{min}) \times ((N_c - i)/(N_c - 1))$
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RU012 – Crear archivo inp de la solución generada.	
Descripción:	Al ejecutar un algoritmo metaheurístico este devuelve una o un conjunto de soluciones. A partir de alguna de estas soluciones se debe crear un archivo inp en el que se vean reflejados los resultados de la solución.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	15/10/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RU013 – Mostrar las soluciones de los algoritmos en la interfaz de usuario.	
Descripción:	Mostrar los resultados de la ejecución del algoritmo, es decir, las variables y los valores objetivos resultantes. Adicionalmente, éstas deben poder ser guardadas en el equipo del usuario. Las variables de decisión se guardarán en un archivo (VAR), mientras que los valores de los objetivos se guardarán en otro (FUN).
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RU014 – Mostrar las características de la red.	
Descripción:	Mostrar las características que posee la red. Esto puede ser realizado cuando se presiona el elemento de la red o agregando algún componente que muestre los elementos que conforman la red.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RU015 – Graficar las soluciones.	
Descripción:	Mostrar en un plano cartesiano las soluciones que se van obteniendo a medida que se ejecuta el algoritmo. Solo considerar hasta 2 objetivos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RU016 – Hacer el programa fácil de extender.	
Descripción:	El programa debe poder fácilmente agregar nuevos algoritmos, operadores y problemas.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	No Funcional

RU017 – Mostrar en la interfaz de usuario los problemas, agrupando los algoritmos que pueden ser utilizados para resolverlos.	
Descripción:	En el menú donde se muestran los problemas que pueden ser resueltos debe aparecer el nombre del problema y en éste se deben agrupar los algoritmos que pueden ser utilizados para resolverlo. EJ: Pumping Schedule > NSGAII > SPA2
Fuente:	Daniel Mora-Meliá
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RU018 – Permitir realizar múltiples simulaciones independientes para resolver el problema multiobjetivo.	
Descripción:	Durante la resolución del problema multiobjetivo se debe poder escoger cuantas veces se corra el algoritmo, independientemente de otra ejecución. La solución final será la Frontera de Pareto del conjunto formado por todas las soluciones generadas a partir de cada ejecución independiente del algoritmo. Este concepto se conoce en algunos framework como Experimentos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RU019 – Guardar los resultados temporales por cada simulación independiente del problema multiobjetivo y generar los archivos al final de todas las simulaciones con los mejores resultados obtenidos.	
Descripción:	Las simulaciones multiobjetivos permiten realizar múltiples simulaciones independientes, entregando cada ejecución como resultado el conjunto de soluciones que conforman su Frontera de Pareto. Sin embargo, una vez se terminan todas las repeticiones y se genera la solución final, se pierden los resultados intermedios de cada repetición. Es por ello, que mientras se van terminando cada simulación independiente, se debe guardar un respaldo de los resultados de cada repetición del algoritmo.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RU020 – Permitir realizar simulaciones hidráulicas utilizando los valores por defectos que vienen en el archivo inp y visualizar los resultados.	
Descripción:	Utilizando los valores que vienen por defecto en el archivo inp se debe poder llevar a cabo la simulación hidráulica de la red. Posteriormente, los resultados podrán ser visualizados por el usuario.
Fuente:	Daniel Mora-Meliá
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RU021 – Agregar el algoritmo multiobjetivo SMPSO.	
Descripción:	Con el fin de comprobar que se pueden acoplar nuevos algoritmos se solicita por parte del usuario que se agregue el algoritmo SMPSO.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RU022 – Agregar el algoritmo multiobjetivo SPA2.	
Descripción:	Con el fin de comprobar que se pueden acoplar nuevos algoritmos se solicita por parte del usuario que se agregue el algoritmo SPA2.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RU023 – Incluir en el dibujo de la red símbolos para diferenciar los elementos que componen la red.	
Descripción:	Se requiere que se agreguen en la representación gráfica de la red símbolos para distinguir los distintos elementos que la componen.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RU024 – Permitir realizar múltiples simulaciones independientes para resolver el problema monoobjetivo.	
Descripción:	Durante la resolución del problema monoobjetivo se debe poder escoger cuantas veces se repetirá el algoritmo, siendo independiente una repetición de la otra. Al finalizar cada repetición se solicita mostrar los resultados de cada repetición.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RU025 – Agregar un menú de configuración.	
Descripción:	Agregar un menú de configuraciones donde poder establecer el tamaño de los puntos, si mostrar o no el gráfico de resultados y limitar el número de resultados retornados por el problema multiobjetivo.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RU026 – Usar en el gráfico, para cada repetición del algoritmo en un Experimento, un color distinto.	
Descripción:	Para facilitar la distinción entre cada repetición de un algoritmo en un experimento se solicita variar que se cambie el color mostrado para cada repetición.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Transable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RU027 – Mostrar una leyenda que pueda ser activada y desactivada con los símbolos mostrados sobre el dibujo de la red.	
Descripción:	Se requiere que en la ventana de representación de la red se muestre una leyenda en la que se presenten los símbolos utilizados por la aplicación para representar cada componente de la red.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RU028 – Mostrar en la ventana de configuración información sobre el problema, como los objetivos, el nombre del algoritmo a utilizar y el nombre del problema que se quiere resolver.	
Descripción:	Al momento de querer realizar una optimización no se entrega suficiente información acerca de los objetivos del problema. Es por esto, que se requiere poder agregar alguna descripción que pueda ser visualizada cuando se abra la ventana de configuración del experimento.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RU029 – Añadir en la ventana de resultados del problema columnas extras que muestren las configuraciones utilizadas con el problema.	
Descripción:	Se solicita que se pueda agregar mas información a la ventana de resultados mostrada cuando se termina una optimización. Se propone usar un mapa con los valores adicionales que se quieran mostrar, en donde la clave sea el nombre de la columna y el valor sea lo mostrado en la celda.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RU030 – Exportar los resultados de los algoritmos aplicados como un Excel.	
Descripción:	Se requiere exportar la table de resultados de la optimización a un archivo Excel.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RU031 – Exportar el gráfico utilizado para mostrar visualmente las soluciones a una imagen (PNG o SVG)	
Descripción:	Se requiere exportar el gráfico de resultados mostrado durante la ejecución de un experimento, ya sea para el problema monoobjetivo como el multiobjetivo. Se requiere idealmente SVG, en caso de que este formato no sea posible se acepta la utilización de PNG.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Transable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RU032 – Permitir indicar valores por defecto a los operadores y a los problemas.	
Descripción:	Se solicita que se puedan ingresar valores por defecto que puedan ser visualizados en la ventana de configuración del problema antes de llevar a cabo la resolución del algoritmo. Los valores por defecto deben ser agregados donde se esperen recibir valores numéricos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

3.2. Requisitos de sistema

En esta sección se presentarán los requisitos de sistema obtenidos a partir de los requisitos de usuarios.

RS001 – Leer red desde el archivo inp.	
Descripción:	Leer un archivo inp desde la aplicación.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	1
Tipo:	Funcional

RS002 – Cargar red dentro del programa.	
Descripción:	Generar una representación de la red en el programa a partir del archivo leído, es decir, crear una jerarquía de clases para mantener la información de los componentes y su configuración leidos desde el archivo inp.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	1
Tipo:	Funcional

RS003 – Implementar Algoritmo Genético.	
Descripción:	Implementar el Algoritmo Genético. La versión del Algoritmo Genético a ser implementado consiste en el Algoritmo Genético Generacional.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RS004 – Implementar el problema multiobjetivo <i>Pipe Optimizing</i>.	
Descripción:	Implementar la clase que lleva a cabo la evaluación de las soluciones generadas por los algoritmos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RS005 – Utilizar Algoritmo Genético para resolver el problema <i>Pipe Optimizing</i> desde el enfoque monoobjetivo.	
Descripción:	Utilizando el Algoritmo Genético y la clase que lleva a cabo la evaluación de las soluciones (La clase <i>Problem</i>) se debe realizar la evaluación y optimización de los objetivos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RS006 – Evaluar soluciones al problema monoobjetivo (<i>Pipe optimizing</i>) usando la librería Epanet.	
Descripción:	Se deben evaluar el cumplimiento de las restricciones establecidas para el problema. La restricción establecida para el problema <i>Pipe Optimizing</i> es cumplir con un nivel de presión mínimo.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RS007 – Implementar NSGAII.	
Descripción:	Implementar el algoritmo NSGAII. El cual es usado para tratar con problemas multiobjetivo.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	01/10/2019
Estado:	Cumple
Incremento:	4
Tipo:	Funcional

RS008 – Implementar el problema multiobjetivo <i>Pumping Schedule</i>.	
Descripción:	Implementar la clase <i>Problem</i> que lleva a cabo la evaluación de las soluciones generadas por los algoritmos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	01/10/2019
Estado:	Cumple
Incremento:	4
Tipo:	Funcional

RS009 – Utilizar el algoritmo NSGAII para resolver el problema <i>Pumping Schedule</i> desde el enfoque multiobjetivo.	
Descripción:	Utilizando el NSGAII y la clase que lleva a cabo la evaluación de las soluciones (La clase <i>Problem</i>) se debe realizar la evaluación y optimización de los objetivos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	01/10/2019
Estado:	Cumple
Incremento:	4
Tipo:	Funcional

RS010 – Evaluar soluciones al problema multiobjetivo (<i>Pumping Schedule</i>) usando la librería Epanet.	
Descripción:	Se deben evaluar el cumplimiento de las restricciones establecidas para el problema.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	01/10/2019
Estado:	Cumple
Incremento:	4
Tipo:	Funcional

RS011 – Visualizar red dentro de un canvas.	
Descripción:	Se debe mostrar la representación gráfica de la red utilizando un Canvas.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS012 – Crear archivo TSV para guardar los valores de los objetivos.	
Descripción:	Se debe crear un archivo TSV (archivo de texto separado por tabuladores) con los objetivos de las soluciones obtenidas.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS013 – Crear archivo TSV para guardar los valores de las variables.	
Descripción:	Se debe crear un archivo TSV (archivo de texto separado por tabuladores) con las variables de las soluciones obtenidas.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	09/09/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS014 – Implementar el operador IntegerSBXCrossover.	
Descripción:	El operador IntegerSBXCrossover es uno de los operadores de cruzamiento. En base a cálculos probabilísticos combina dos soluciones para crear unas dos nuevas soluciones hijas.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RS015 – Implementar el operador IntegerSinglePointCrossover.	
Descripción:	El operador IntegerSinglePointCrossover es un operador de cruzamiento. Viendo la solución como un vector, este operador toma dos soluciones y elige un punto a partir del cual los valores de una solución se intercambiarán con los valores de otra solución. Este operador usa una probabilidad de cruzamiento y solamente realiza el intercambio de los valores en la solución cuando un número generado aleatoriamente es menor que la probabilidad de cruzamiento.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RS016 – Implementar el operador IntegerPolynomialMutation.	
Descripción:	El operador IntegerPolynomialMutation es un operador de mutación. Este operador de mutación usa cálculos probabilísticos para mutar algunos variables de decisión que forman parte de la solución.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RS017 – Implementar el operador IntegerSimpleRandomMutation.	
Descripción:	El operador IntegerSimpleRandomMutation es un operador de mutación. Este operador muta una variable de decisión cuando un número generado aleatoriamente es menor que la probabilidad de mutación establecida. El operador recorre cada variable de decisión realizando lo descrito anteriormente. La mutación realizada por este operador consiste en cambiar el valor de la variable de decisión por otro valor aleatorio.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RS018 – Implementar el operador IntegerRangeRandomMutation.	
Descripción:	<p>El operador IntegerRangeRandomMutation es un operador de mutación. Este operador muta una variable de decisión cuando un número generado aleatoriamente es menor que la probabilidad de mutación establecida. El operador recorre cada variable de decisión realizando lo descrito anteriormente. La mutación realizada por este operador consiste en cambiar el valor de la variable de decisión por otro valor aleatorio que se encuentre entre un rango establecido.</p> <p>Ejemplo:</p> <p>Variable de decisión: 3 Rango: 2 La variable de decisión después de aplicado el operador puede tomar un valor entre [1, 5].</p>
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RS019 – Implementar el operador UniformSelection.	
Descripción:	<p>El operador UniformSelection [2] es un operador de selección. Este operador de selección ordena la población y asigna una probabilidad máxima y mínima a la mejor y peor solución, respectivamente. A las soluciones que se encuentran entre la mejor y la peor solución se le asigna una probabilidad entre el máximo y mínimo obtenido anteriormente. Si la probabilidad de la solución es mayor a 1.5 entonces la solución se duplica en la nueva población. Si la probabilidad está entre 0.5 y 1.5, entonces en la nueva población se agrega la solución solo una vez. Las soluciones cuya probabilidad es menor que 0.5 no aparecen en la nueva población.</p> <p>La probabilidad máxima puede ser calculada como $p_{max} = \beta/N_c$ mientras que la probabilidad mínima se calcula de acuerdo a $p_{min} = (2-\beta)/N_c$. Donde β es un número entre 1.5 y 2, y N_c es el número de soluciones presentes en el conjunto sobre el que se realizará la selección.</p> <p>La probabilidad de las soluciones intermedias se calcula de acuerdo a</p> $p_i = p_{min} + (p_{max} - p_{min}) \times ((N_c - i)/(N_c - 1))$
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	14/10/2019
Estado:	Cumple
Incremento:	2
Tipo:	Funcional

RS020 – Aplicar una solución al objeto que representa la red.	
Descripción:	Cuando el algoritmo haya generado soluciones para el problema, se debe poder seleccionar una de ellas y aplicarlas sobre un objeto <i>Network</i> el cual podrá ser exportado posteriormente a un archivo .inp. Aplicar la solución sobre un objeto <i>Network</i> significa tomar los valores de las variables de decisión y modificar las configuraciones de la red (instancia de <i>Network</i>) donde corresponda..
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	15/10/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS021 – Exportar el objeto red a un archivo .inp.	
Descripción:	Se debe exportar el objeto <i>Network</i> a un archivo con extensión .inp, de acuerdo al formato establecido por Epanet. El archivo .inp generado debe poder ser cargado en el programa de simulación Epanet.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	15/10/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS022 – Crear la pestaña que permite visualizar los resultados.	
Descripción:	Cuando el experimento haya terminado su ejecución se debe crear un nuevo componente para mostrar los resultados.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS023 – Mostrar los resultados de la optimización.	
Descripción:	En la pestaña de resultados se deben mostrar los resultados de la optimización.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS024 – Implementar un componente que permita mostrar los elementos que componen la red.

Descripción:	El componente debe estar filtrado por el tipo de elemento que compone la red.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS025 – Implementar componente que permitan mostrar las características de un elemento seleccionado de la red.

Descripción:	Se debe crear una ventana o un componente en el que se muestren la configuración de los elementos de la red. Este componente debe actualizarse cada vez que se seleccione un nuevo elemento, ya sea desde la lista o desde la representación gráfica de la red.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS026 – Permitir seleccionar en la lista de elementos de la red el componente del que se quieren mostrar sus características.	
Descripción:	En la lista de elementos de la red, al hacer doble click, se debe mostrar el componente que muestra las configuraciones que están establecidas para el elemento seleccionado.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS027 – Permitir seleccionar del dibujo de la red el componente del que se van a mostrar sus características.	
Descripción:	Al hacer doble click sobre un elemento, en el canvas que representa la red, se debe abrir el componente para ver la configuración que tiene establecida dicho elemento.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS028 – Implementar un componente que muestre un plano cartesiano.	
Descripción:	Este componente gráfico debe tener un plano cartesiano. El plano solo puede ser ocupado cuando la optimización es uno o dos objetivos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS029 – Mostrar las soluciones del experimento monoobjetivo en el plano cartesiano.	
Descripción:	El componente de gráficos debe permitir agregar el valor de la solución al plano cartesiano. El plano cartesiano tendrá en el eje y el objetivo, y en el eje x el número de generaciones.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS030 – Mostrar las soluciones del experimento multiobjetivo en el plano cartesiano.	
Descripción:	El componente de gráficos debe permitir agregar el valor de la solución al plano cartesiano. El plano cartesiano usará en el eje x el objetivo 1, mientras que el objetivo 2 estará en el eje y.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Transable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	Funcional

RS031 – Implementar una jerarquía de clases que a través de la implementación de interfaces se pueda extender los algoritmos.	
Descripción:	Implementar una jerarquía de clases que permita agregar nuevos algoritmos o modificar los ya existentes fácilmente. Entendiendo por fácilmente, que solo se necesita implementar una interfaz para agregar un nuevo algoritmo o extender alguna clase de un algoritmo ya existente para modificar su comportamiento.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	No Funcional

RS032 – Implementar una jerarquía de clases que a través de la implementación de interfaces se pueda extender los problemas.	
Descripción:	Implementar una jerarquía de clases que permita agregar nuevos problemas fácilmente. Entendiendo por fácilmente, que solo se necesita implementar una interfaz para agregar un nuevo problema o extender alguna clase de un problema ya existente para modificar su comportamiento.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	No Funcional

RS033 – Implementar una jerarquía de clases que a través de la implementación de interfaces se pueda extender los operadores.	
Descripción:	Implementar una jerarquía de clases que permita agregar nuevos operadores o modificar los ya existentes fácilmente. Entendiendo por fácilmente, que solo se necesita implementar una interfaz para agregar un nuevo operador o extender alguna clase de un operador ya existente para modificar su comportamiento.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	30/11/2019
Estado:	Cumple
Incremento:	3
Tipo:	No Funcional

RS034 – Agrupar para cada problema mostrado en el menú de la interfaz de usuario los algoritmos que pueden ser usados.	
Descripción:	Dentro del menú de optimización cada problema será un nuevo menú, los cuales indicaran los algoritmos que pueden ser usados.
Fuente:	Daniel Mora-Meliá
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RS035 – Implementar mecanismo que permita realizar múltiples repeticiones del mismo algoritmo para un problema específico desde el enfoque multiobjetivo.	
Descripción:	Durante la resolución del problema multiobjetivo se debe poder escoger cuantas veces se ejecutará el algoritmo de manera independiente de otras ejecuciones. La solución final será la Frontera de Pareto del conjunto formado por todas las soluciones generadas a partir de cada ejecución independiente del algoritmo. Este concepto se conoce en algunos framework como Experimentos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RS036 – Mantener los resultados de cada repetición de un algoritmo cuando se resuelve un problema multiobjetivo.	
Descripción:	Al resolver un problema multiobjetivo se deben mantener los resultados intermedios generados cuando se termina cada repetición del algoritmo que conforma el Experimento.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RS037 – Guardar los resultados almacenados de cada repetición del algoritmo multiobjetivo.	
Descripción:	Los resultados intermedios generados por cada ejecución independiente del algoritmo metaheurístico multiobjetivo, deben ser guardados en una carpeta indicada por el usuario.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RS038 – A partir de los resultados de cada repetición del algoritmo, para el problema multiobjetivo obtener las mejores soluciones.	
Descripción:	A partir de los resultados intermedios generados por cada repetición del algoritmo multiobjetivo se deben obtener las soluciones no dominadas, es decir, aquellas que conforman la Frontera de Pareto.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RS039 – Guardar las mejores soluciones obtenidas a partir de cada una de las repeticiones del algoritmo.	
Descripción:	Guardar las soluciones no dominadas obtenidas al unir los resultados de cada repetición del algoritmo dentro de un Experimento.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RS040 – Realizar la simulación hidráulica utilizando los valores por defecto de la red que vienen en el archivo inp.	
Descripción:	Se debe poder realizar la simulación hidráulica utilizando únicamente los valores preconfigurados en el archivo inp.
Fuente:	Daniel Mora-Meliá
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RS041 – Obtener los resultados de la simulación hidráulica desde el simulador.	
Descripción:	A medida que se van realizando la simulación hidráulica se deben recuperar los resultados del simulador y almacenarlos en memoria para posteriormente ser presentados al usuario.
Fuente:	Daniel Mora-Meliá
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RS042 – Mostrar los resultados de la simulación hidráulica en la interfaz de usuario.	
Descripción:	Mostrar al usuario los resultados obtenidos al realizar la simulación.
Fuente:	Daniel Mora-Meliá
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	27/01/2020
Estado:	Cumple
Incremento:	5
Tipo:	Funcional

RS043 – Implementar el algoritmo SMPSO.	
Descripción:	Agregar el algoritmo SMPSO (Multiobjetivo) y utilizarlo para resolver el problema Pumping Schedule.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS044 – Implementar el algoritmo SPA2.	
Descripción:	Agregar el algoritmo SPA2 (Multiobjetivo) y utilizarlo para resolver el problema Pumping Schedule.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS045 – Especificar los símbolos a utilizar para cada componente de la red.	
Descripción:	Establecer símbolos para representar los elementos de la red. Se pueden usar los mismos que el software de simulación hidráulica Epanet.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS046 – Agregar los símbolos de los componentes cuando se muestra la red.	
Descripción:	Agregar sobre cada elemento mostrado en la representación de la red el símbolo que lo representa y que permite distinguirlo de otros elementos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS047 – Implementar mecanismo que permita realizar múltiples repeticiones del mismo algoritmo para un problema específico desde el enfoque monoobjetivo.	
Descripción:	Durante la resolución del problema monoobjetivo se debe poder escoger cuantas veces se repetirá el algoritmo, siendo independiente una repetición de la otra. Al finalizar cada repetición se solicita mostrar los resultados de cada repetición. Así como se hace para problemas multiobjetivos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS048 – Crear una interfaz de usuario para mostrar las configuraciones.	
Descripción:	Implementar una interfaz de usuario para presentar ciertas opciones que el usuario puede configurar al utilizar la aplicación.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS049 – Establecer los valores de la aplicación que se permitirá que el usuario configure.	
Descripción:	Establecer cuáles serán los valores permitidos para que el usuario configure en la ventana de configuración.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	No Funcional

RS050 – Mostrar al usuario la ventana de configuración.	
Descripción:	Agregar la funcionalidad para abrir la ventana de configuración cuando el usuario la requiera.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS051 – Aplicar al sistema las configuraciones establecidas en la ventana de configuraciones.	
Descripción:	Hacer efectivas las configuraciones configuradas en la ventana de configuración en el programa.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS052 – Escoger una paleta de colores a partir de la cual elegir el color usado para mostrar cada iteración.	
Descripción:	Establecer una paleta de colores entre las que poder elegir el color a ser usado por cada iteración dentro de un experimento.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Transable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	No Funcional

RS053 – Asignar a cada solución de una misma iteración un color dentro de la paleta de colores.	
Descripción:	Implementar la funcionalidad para que cada iteración a ser mostrada en el gráfico de resultados tome un color diferente.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Transable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS054 – Mostrar los símbolos usados en la representación de la red y que significan cada uno de ellos.	
Descripción:	Implementar código para que se muestre una leyenda con los símbolos de la red existentes y a qué pertenecen.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS055 – Agregar una opción que permita activar y desactivar la leyenda.	
Descripción:	Agregar en el menú de configuraciones la posibilidad de activar y desactivar la leyenda de símbolos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS056 – Implementar un mecanismo que permite indicar una descripción del problema a resolver.	
Descripción:	Implementar alguna manera de poder agregar una descripción al problema a tratar que debe ser mostrada en la interfaz de configuración del problema como una pestaña. Pueden ser usadas las mismas anotaciones para este fin.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS057 – Mostrar en la ventana de configuración información del problema a resolver.	
Descripción:	Modificar la ventana de configuración del problema para que se muestre la descripción del problema escogido.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS058 – Implementar un mecanismo para indicar el valor de las columnas adicionales que quieren ser mostrados en la interfaz de resultados.	
Descripción:	Implementar código para añadir información adicional a la ventana de resultados. Esta información adicional será especificada por el usuario que implemente el problema. Añadir esta información debe ser opcional. Si esta información no se añade entonces mostrar los resultados y objetivos.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS059 – Agregar los datos adicionales que desean ser mostrados en la ventana de resultados.	
Descripción:	Mostrar los parámetros adicionales en la ventana de resultados.
Fuente:	Jimmy Gutiérrez
Prioridad:	Alta
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS060 – Exportar todos los datos de la ventana de resultados a un archivo Excel.

Descripción:	Implementar código para exportar los valores presentes en la ventana de resultados a un archivo Excel.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS061 – Configurar la carpeta en la que se guardarán la imagen del gráfico.

Descripción:	Mostrar una ventana para que el usuario seleccione donde quiere guardar la imagen del gráfico.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS062 – Guardar el gráfico de resultados en el equipo del usuario en formato PNG.

Descripción:	Guardar el gráfico en la carpeta seleccionada por el usuario.
Fuente:	Jimmy Gutiérrez
Prioridad:	Baja
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS063 – Agregar un mecanismo para ingresar valores por defecto para los operadores.

Descripción:	Implementar funcionalidad para agregar valores por defecto a los parámetros que pueden ser configurados desde la ventana de configuración del problema. Estos valores por defecto deben ser para los parámetros numéricos (int o double).
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

RS064 – Mostrar en la ventana de configuración de los problemas los valores por defecto.	
Descripción:	Al abrir la ventana de configuración de problemas se deben mostrar los valores establecidos por defecto para cada campo.
Fuente:	Jimmy Gutiérrez
Prioridad:	Moderada
Estabilidad:	Intransable
Fecha de actualización:	13/05/2020
Estado:	Cumple
Incremento:	6
Tipo:	Funcional

3.3. Matriz de Trazado Requisitos de Usuario vs. Requisitos de Software

La matriz de trazabilidad de los requisitos de usuario y de sistema que se presenta a continuación permite ver la relación y dependencia que un requisito de sistema tiene con los requisitos de usuario.

R023	R024	R025	R026	R027	R028	R029	R030	R031	R032
				x					
		x x x x							
	x x								
		x x							
			x x						
		x x							
			x						
x x									

Figura 3.1: Matriz de requisito de usuario versus requisitos de sistema.

B. Documento de diseño



UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Documento de diseño

**HERRAMIENTA PARA LA OPTIMIZACIÓN DE REDES DE
DISTRIBUCIÓN DE AGUA POTABLE**

TABLA DE CONTENIDOS

	página
Tabla de Contenidos	
Índice de Figuras	ii
1. Introducción	4
1.1. Propósito del sistema	4
1.2. Alcance del proyecto	4
1.3. Descripción general	5
1.4. Definiciones, siglas y abreviaturas	6
1.5. Referencias	7
2. Diseño arquitectónico	8
2.1. Arquitectura Física	8
2.2. Arquitectura lógica	9
3. Diseño detallado	11
3.1. Diseño detallado de módulos	11
3.2. Diseño de estructura de sistema	12
3.3. Operación del sistema	15
4. Diseño de interfaces	21
4.1. Interfaz de inicio	21
4.2. Interfaz de configuración y descripción del problema	22
4.3. Interfaz de ejecución del experimento	24
4.4. Interfaz de gráficos	27
4.5. Interfaz de resultados	28
4.6. Interfaz de visualización de configuraciones de la red	29
4.7. Interfaz de ejecución de una red	30
5. Detalles de implementación	32
5.1. Uso de <i>Java Annotation</i> y <i>Java Reflection</i>	33
5.1.1. Anotaciones para los operadores	33

5.1.2. Anotaciones para los objetos que heredan la interfaz <i>Registrable</i>	34
5.2. Interfaz Registrable	41

ÍNDICE DE FIGURAS

	página
2.1. Arquitectura física	8
2.2. Arquitectura lógica	9
3.1. Diagrama de componentes	11
3.2. Diagrama de clases general del sistema	13
3.3. Diagrama de clase de la red	14
3.4. Diagrama de clases del componente <i>metaheuristic</i>	15
3.5. Diagrama de secuencia de la carga y visualización de la red	17
3.6. Diagrama de secuencia de la simulación de la red usando los valores del archivo de configuración de red (inp)	18
3.7. Diagrama de secuencia de la optimización	19
3.8. Diagrama de actividades	20
4.1. Esquema de interfaz de inicio del sistema	22
4.2. Interfaz de descripción del problema	23
4.3. Interfaz de configuración del problema	24
4.4. Interfaz de ejecución del experimento monoobjetivo	25
4.5. Interfaz de ejecución del experimento multiobjetivo	26
4.6. Interfaz de gráfico de soluciones para problemas monoobjetivos	27
4.7. Interfaz de gráfico de soluciones para problemas multiobjetivos	28
4.8. Interfaz de resultados de la optimización	29
4.9. Interfaz de visualización de la configuración de la red	30
4.10. Interfaz de resultados de ejecución de simulación hidráulica	31
5.1. Constructor de un solo parámetro	33
5.2. Constructor de un solo parámetro	33
5.3. Interfaz para configurar el operador <i>UniformSelection</i>	34
5.4. Interfaz para configurar el operador <i>IntegerPolynomialMutation</i>	34
5.5. Constructor de clase que hereda de registrable y sus metadatos para cada parámetro	35
5.6. Menú de problemas	36
5.7. Componente <i>ComboBox</i> para configurar los operadores	37

5.8. <i>ComboBox</i> expandido para configurar el operador.	38
5.9. Apartado para configurar el parámetro de tipo <i>File</i>	39
5.10. <i>TextField</i> presente cuando esta la anotación <i>@NumberInput</i>	39
5.11. Apartado para <i>NumberToggleInput</i> con el mismo GroupID.	40

1. Introducción

En este capítulo se presenta de manera general el propósito del sistema, los alcances del proyecto, una descripción de la herramienta a desarrollar, una series de definiciones que son necesarias conocer para la elaboración de este documento.

1.1. Propósito del sistema

El proyecto consiste en el desarrollo de un sistema que permita simular y buscar soluciones a problemas presentes en las redes de distribución de agua potable haciendo uso de algoritmos metaheurísticos. Además, este sistema será desarrollado de tal forma que sea escalable con el fin de que otros desarrollos o investigadores sean capaces de extender su funcionalidad fácilmente agregando nuevos algoritmos metaheurísticos, operadores y problemas.

1.2. Alcance del proyecto

Al final del período de desarrollo la herramienta debe contar con las siguientes prestaciones.

- El sistema permite la carga y la visualización de la red gráficamente.
- El sistema implementa por defecto dos algoritmos. Uno de ellos monoobjetivo y el otro multiobjetivo. El algoritmo monoobjetivo corresponde al Algoritmo Genético (GA) mientras que el multiobjetivo a *Non-Dominated Sorting Genetic Algorithm II* (NSGAII).

- El sistema proporciona dos problemas ya implementados uno monoobjetivo y el otro multiobjetivo. El Algoritmo Genético se utiliza para generar soluciones para el problema monoobjetivo con el fin de optimizar el costo de inversión de las tuberías. Por otro lado, el algoritmo NSGAII aborda el problema multiobjetivo con el objetivo de optimizar los costos energéticos y los costos de mantenimiento de los equipos de bombeo.
- El sistema permite visualizar y guardar las soluciones de los algoritmos en un archivo.
- El sistema permite que el usuario agregue nuevos algoritmos, operadores o problemas sin tener que modificar la interfaz de usuario.

Este proyecto no contempla la creación de la red por lo que estas deberán ser ingresadas como entradas al programa. La creación de las redes puede realizarse a través de EPANET. Además, esta herramienta únicamente podrá ser ocupada en equipos cuyo sistema operativo sea Windows de 64bits debido a que se realizan llamadas a librerías nativas las cuales no son multiplataforma. Adicionalmente, el equipo debe contar con Java 1.8.

1.3. Descripción general

El proyecto es un medio que permite la implementación de algoritmos metaheurísticos para abordar problemas asociados a redes de agua potable.

Inicialmente se incorporará la presencia de dos metaheurísticas las cuales son el Algoritmo Genético y el algoritmo NSGAII. Cada uno de estos algoritmos está enfocado en la búsqueda de soluciones para un problema determinado. GA está enfocado en la búsqueda de soluciones para optimizar los de costos de inversión en la construcción de una red desde el enfoque monoobjetivo. En cambio, el algoritmo NSGAII busca soluciones para el problema multiobjetivo, cuyos objetivos son optimizar el costo de operación junto con el costo de mantenimiento de los equipos de bombeo (*Pumping Schedule*).

1.4. Definiciones, siglas y abreviaturas

Metaheurísticas: Algoritmos que permiten resolver un amplio rango de problemas de optimización empleando técnicas con algún grado de aleatoriedad para encontrar soluciones a un problema. Estos algoritmos no garantizan que la solución encontrada sea la óptima, pero permiten obtener generalmente aproximaciones a esta [9, 1, 5].

Genetic Algorithm (GA): Estrategia de búsqueda de soluciones basada en la teoría de la evolución de Darwin. Para realizar esto, el algoritmo parte desde un conjunto de soluciones denominada población he iterativamente, lleva a cabo un proceso de reproducción, generando nuevas soluciones [4].

Non-dominated Sorting Genetic Algorithm (NSGAI): Algoritmo que utiliza el cruzamiento, mutación y reproducción para encontrar un conjunto de soluciones óptimas a problemas que cuentan con más de un objetivo [3].

Java Reflection: Característica de java que permite que un programa se auto examine. Esta característica está disponible a través de la *Java Reflection API*, la cual cuenta con métodos para obtener los *meta-object* de las clases, métodos, constructores, campos o parámetros. Esta API también permite crear nuevos objetos cuyo tipo era desconocido al momento de compilar el programa [2].

Java Annotation: Característica de java para agregar metadatos a elementos de java (clases, métodos, parámetros, etc.) [7]. Las anotaciones no tienen efecto directo sobre el código, pero cuando son usadas junto con otras herramientas pueden llegar a ser muy útiles. Estas herramientas pueden analizar estas anotaciones y realizar acciones en base a estas, por ejemplo, generar archivos adicionales como clases de java, archivos XML, entre otras. Adicionalmente, las anotaciones también pueden ser analizadas durante la ejecución del programa vía *Java Reflection*, para crear objetos cuyo tipo no conocemos en tiempo de compilación.

GUI: Graphical User Interface.

1.5. Referencias

- [1] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.
- [2] Mathias Braux and Jacques Noyé. Towards partially evaluating reflection in Java. *ACM SIGPLAN Notices*, 34(11):2–11, 1999.
- [3] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 2002.
- [4] Dorothea Heiss-Czedik. An introduction to genetic algorithms. *Artificial Life*, 3(1):63–65, 1997.
- [5] Sean Luke. *Essentials of metaheuristics*. Lulu, second edition, 2013.
- [6] Antonio J. Nebro, Juan J. Durillo, and Matthieu Vergne. Redesigning the jMetal multi-objective optimization framework. In *GECCO 2015 - Companion Publication of the 2015 Genetic and Evolutionary Computation Conference*, 2015.
- [7] Henrique Rocha and Marco Tulio Valente. How annotations are used in Java: An empirical study. *SEKE 2011 - Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering*, (June 2014):426–431, 2011.
- [8] Artem Syromiatnikov and Danny Weyns. A journey through the land of model-view-design patterns. *Proceedings - Working IEEE/IFIP Conference on Software Architecture 2014, WICSA 2014*, pages 21–30, 2014.
- [9] Xin She Yang. Metaheuristic optimization. *Scholarpedia*, 6(2011):11472, 2015.

2. Diseño arquitectónico

En este capítulo se presenta la arquitectura física y lógica del sistema. Además, se hace mención a los diversos módulos que componen a esta última.

2.1. Arquitectura Física

La arquitectura física del software a desarrollar consiste en la arquitectura monolítica. Puesto que es un programa de escritorio cuya persistencia de datos se realiza en el mismo equipo del usuario que está ejecutando la aplicación y no necesita interacción con otros sistemas o equipos. En la Figura 2.1 se puede ver una imagen que representa la arquitectura.

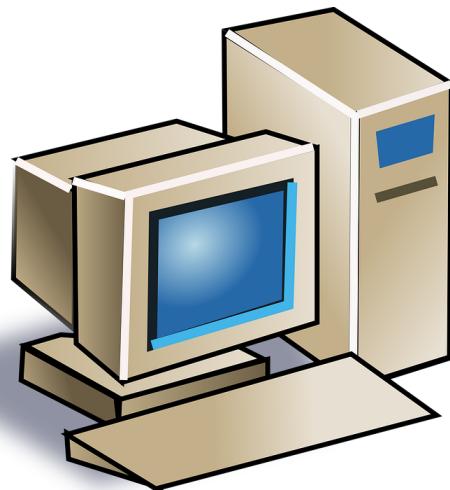


Figura 2.1: Arquitectura física.

2.2. Arquitectura lógica

La arquitectura lógica definida para la herramienta corresponde al patrón Modelo-Vista-Controlador [8]. La elección de este patrón se debe a que permite la separación de la lógica de negocio y la vista presentada al usuario logrando de esta manera una aplicación altamente mantenible y con una mejor escalabilidad. El diagrama que representa la arquitectura usada para el desarrollo se puede ver en la Figura 2.2.

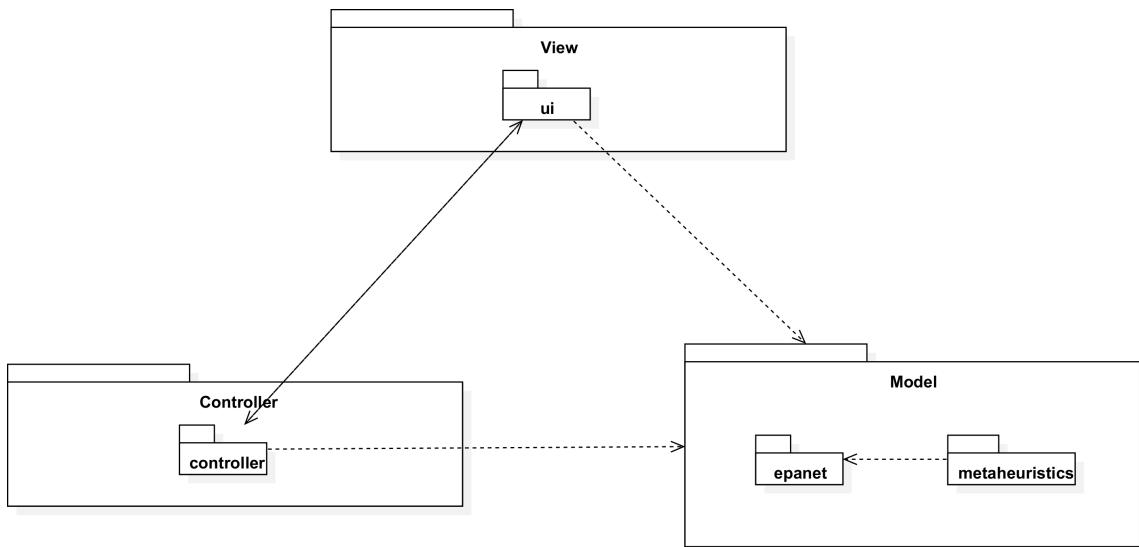


Figura 2.2: Arquitectura lógica.

En el diagrama presentado en la Figura 2.2 se presencia la división de la aplicación en tres capas. La capa Vista se encarga de la interacción con el usuario y de la interfaz de usuario. La capa Controlador se encarga de responder a los eventos de los usuarios y solicitar información a la capa de modelo. La capa Modelo contiene la información y la lógica fundamental de la aplicación en un formato adecuado para interactuar con las demás capas. Esta última capa, también se encarga de la ejecución de los algoritmos y la interacción con la librería nativa Epanet Toolkit.

La capa Modelo se divide principalmente en dos componentes. Estos componentes son:

Componente Metaheurísticas (*Metaheuristics*): Modulo que contiene los algoritmos, operadores, los tipos de solución permitidos y los problemas que serán

abarcados durante el desarrollo del proyecto.

Componente Hidráulico (EPANET): El módulo hidráulico posee las clases necesarias para cargar los archivos de red (inp) y generar una representación de ellos a través de una serie de clases. Este módulo también se encarga de guardar la representación de una red ya modificada en un archivo inp para que pueda ser usado por el programa EPANET. Las simulaciones hidráulicas serán realizadas usando la librería epajava. Esta librería realiza las llamadas nativas a la Epanet Toolkit. La librería epajava puede ser descargada en el repositorio de git ubicado en <https://github.com/jhawanet/epajava> .

3. Diseño detallado

Durante esta sección se presenta el diseño de la aplicación de manera más detallada, acompañado junto con una series de diagramas que describen la funcionalidad o algunas operaciones que realiza el sistema.

3.1. Diseño detallado de módulos

Los componentes que forman la aplicación son el componente de la GUI, el de los Controladores, el componente de Epanet y el componente Metaheurísticas. La relación entre los componentes puede ser apreciada en la Figura 3.1.

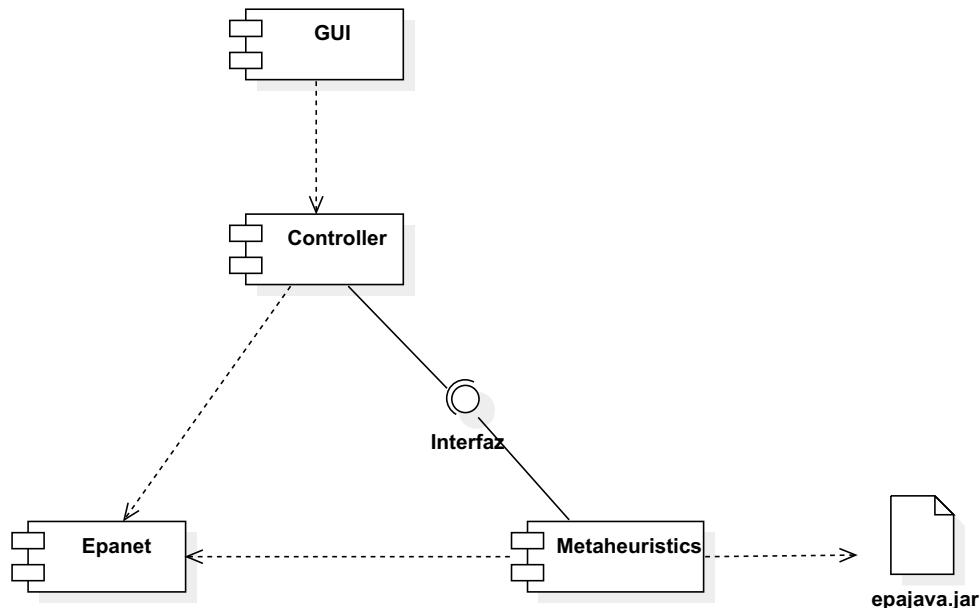


Figura 3.1: Diagrama de componentes.

A continuación, se describe cada uno de los componentes presentados en la Figura 3.1:

GUI (Graphics User Interface) Este componente está encargado de presentar todas las vistas. Entre las vistas se encuentra la ventana principal de la aplicación, la ventana de configuración de para un problema, la ventana de ejecución de algoritmos y la ventana de resultados.

Controller El controlador se encarga de manejar los eventos generados por la *GUI*. Generalmente la relación es uno a uno, es decir, por cada interfaz de usuario hay un controlador. Debido a que la interfaz de usuario está formada por varios componentes gráficos, se da el caso en que cada uno de éstos puede tener su propio controlador.

Epanet Este componente esta encargada de la lectura y la escritura de archivos de configuración de red (Llamado desde ahora como archivo inp). Este componente cuenta también con clases que representan una red cargada desde un inp. Estas clases permitirán editar, durante la ejecución del programa, algunas configuraciones de la red. Posteriormente, estas configuraciones podrán ser persistidas generando un nuevo archivo de configuración de red.

Metaheuristics Este componente contiene los algoritmos metaheurísticos, así como los problemas y los operadores que pueden ser ocupados por los algoritmos.

epajava.jar Esta es una librería para la simulación de las redes de agua potable. Esta librería permite realizar llamadas nativas a la librería de Epanet. Estas llamadas nativas se hacen a través de la librería *JNA* (*Java Native Access*). Para ocupar la librería, esta requiere que se indique el archivo de descripción de la red (Archivo inp).

3.2. Diseño de estructura de sistema

En la Figura 3.2 se muestra un diagrama general de los componentes, sus clases más importantes y su interacción. En la Figura 3.2 se muestra un diagrama general de los componentes, sus clases más importantes y su interacción.

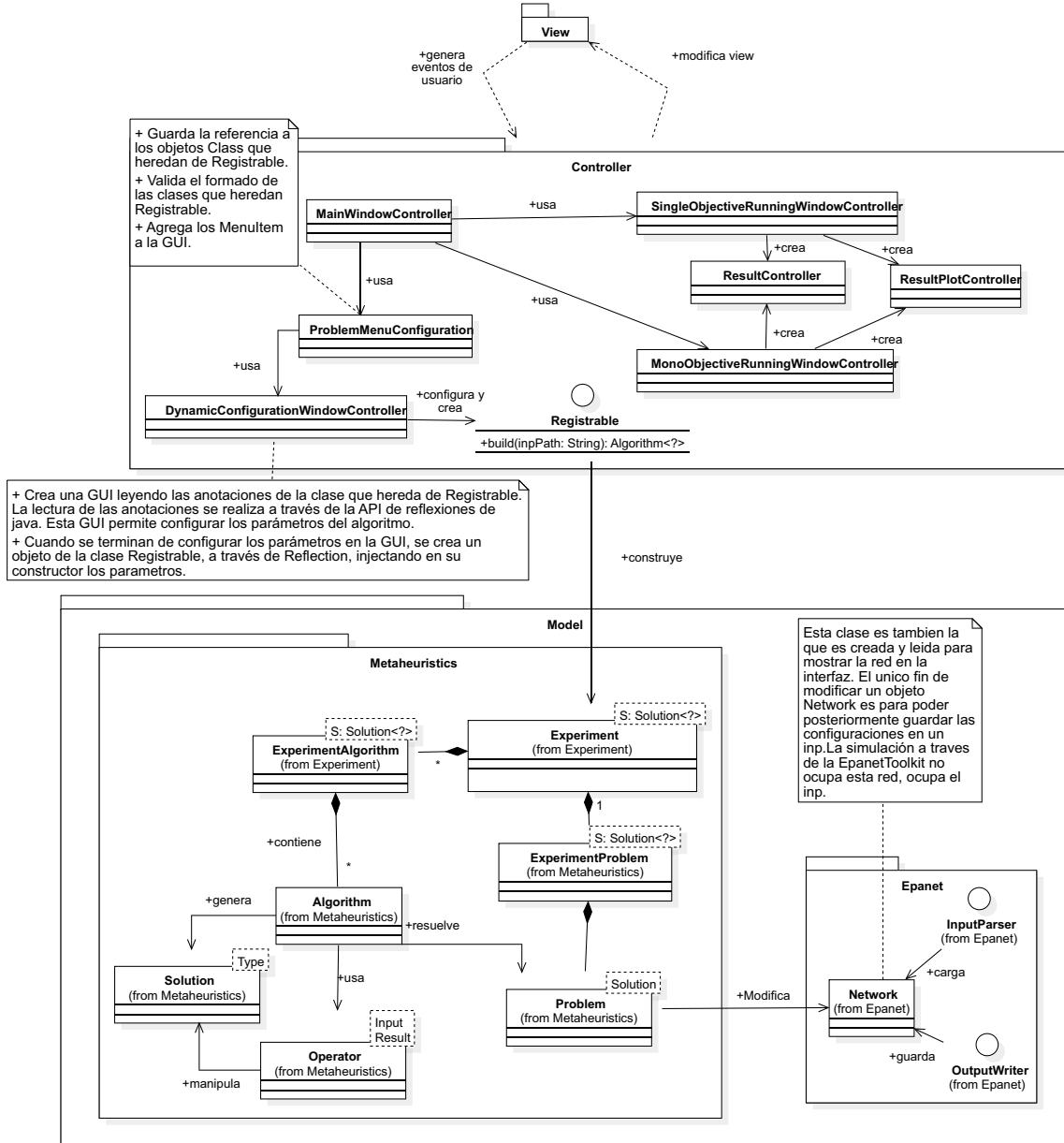


Figura 3.2: Diagrama de clases general del sistema.

La Figura 3.3 corresponde a un diagrama de clases más detallado del componente Epanet.

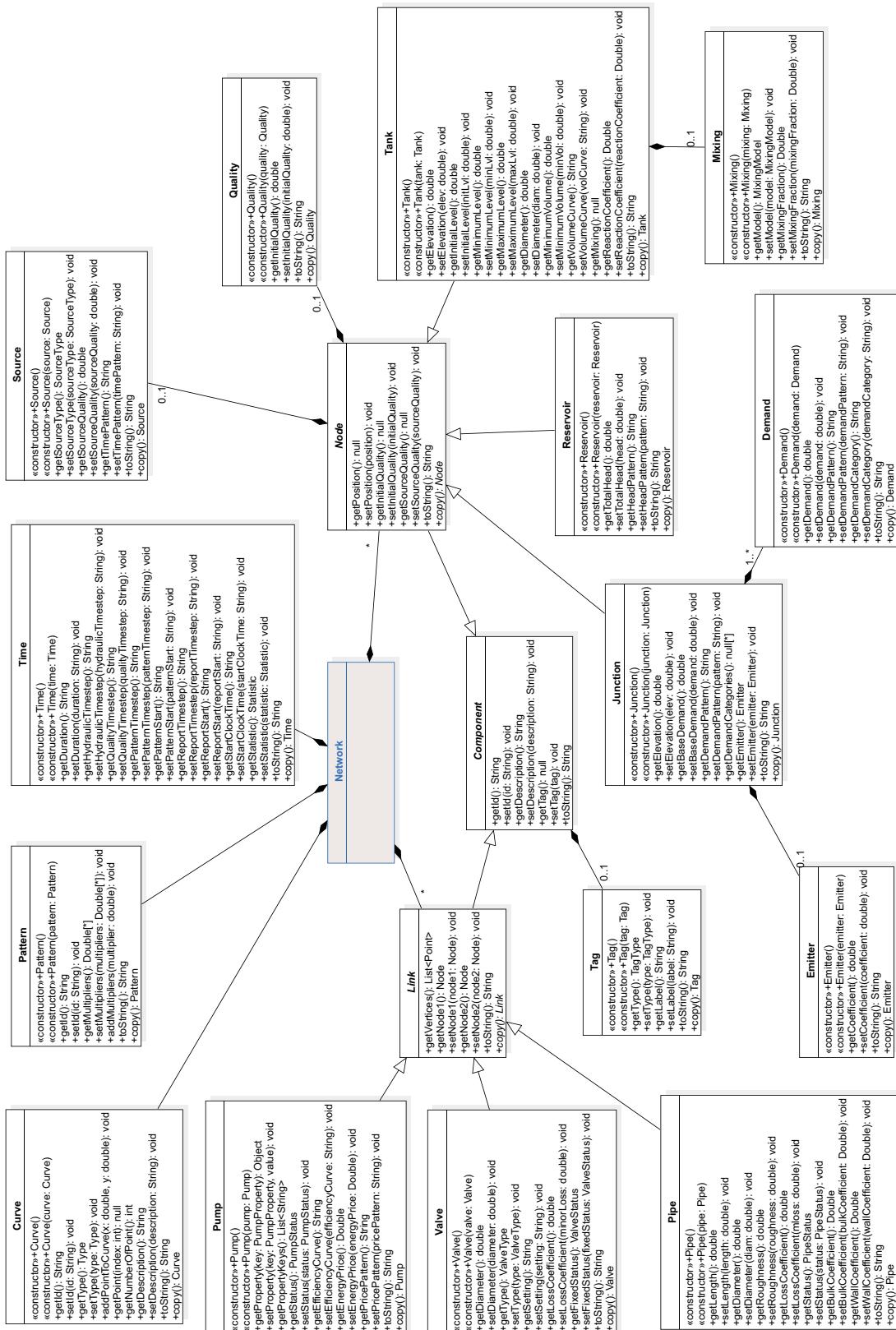


Figura 3.3: Diagrama de clase de la red.

En la Figura 3.4, se presenta el diagrama de clases del componente metaheurístico. Este fue tomado de [6] y adaptado para ser usado por nuestra aplicación.

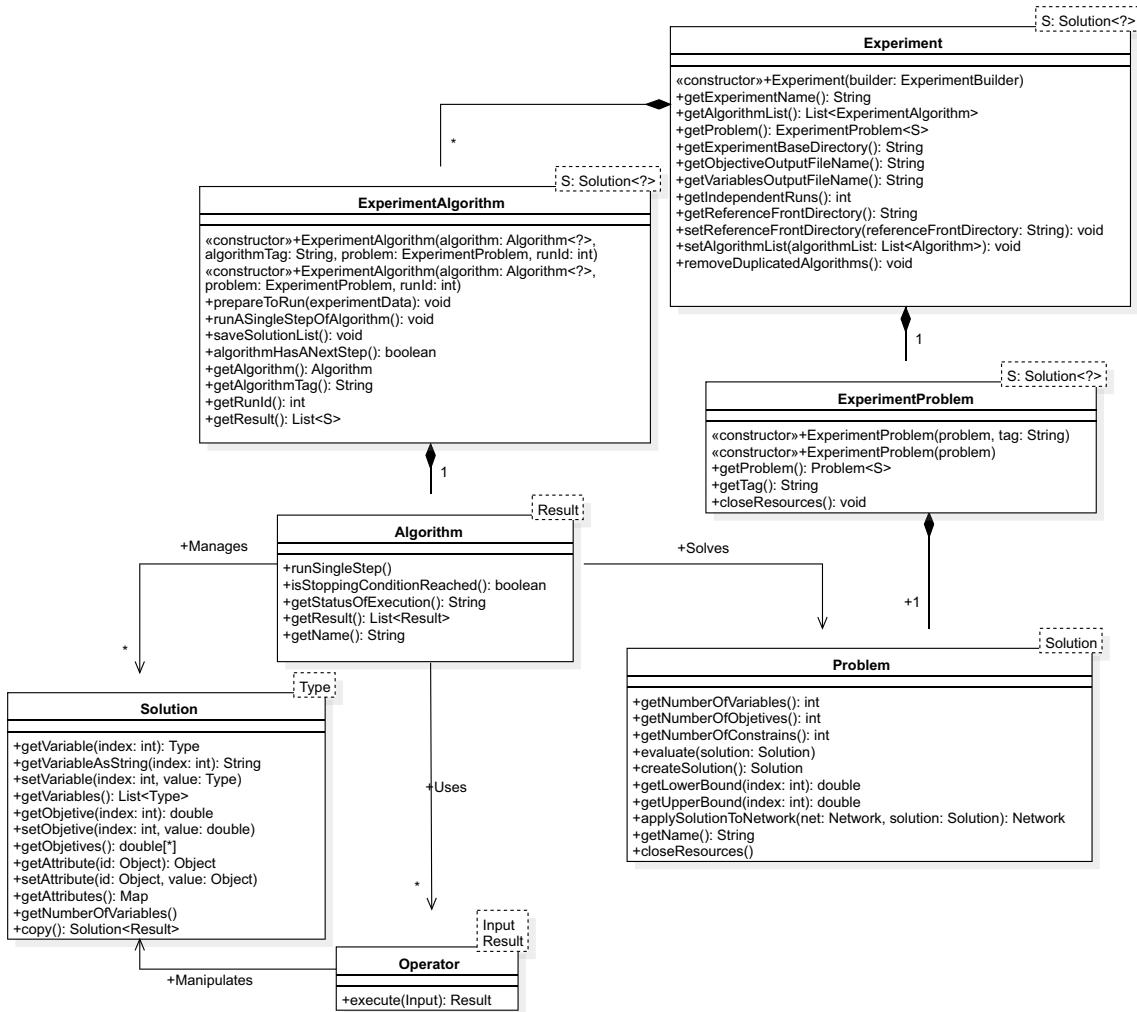


Figura 3.4: Diagrama de clases del componente *metaheuristic*.

3.3. Operación del sistema

A continuación, se presentan una serie de diagramas de secuencia que describen las interacciones entre las clases para ciertas funcionalidades.

En la Figura 3.5, se presenta la secuencia de tareas que realiza la aplicación desde que es abierta hasta que se visualiza la red.

Luego, en la Figura 3.6, se muestra la serie de acciones realizadas para realizar la simulación hidráulica utilizando los valores por defecto del archivo de red.

Después, en la 3.7, se puede observar la interacción entre las clases de la aplicación para poder llevar a cabo la resolución de un problema, sea este monoobjetivo o multiobjetivo.

Finalmente, en la Figura 3.8 se muestra un diagrama de actividades de las operaciones que se pueden llevar a cabo con la aplicación. En este diagrama de actividades, los nodos amarillos corresponde a la parte del proceso en que se utiliza *Java Reflection API* y *Java Annotation*, los cuales son presentados en el capítulo 5.

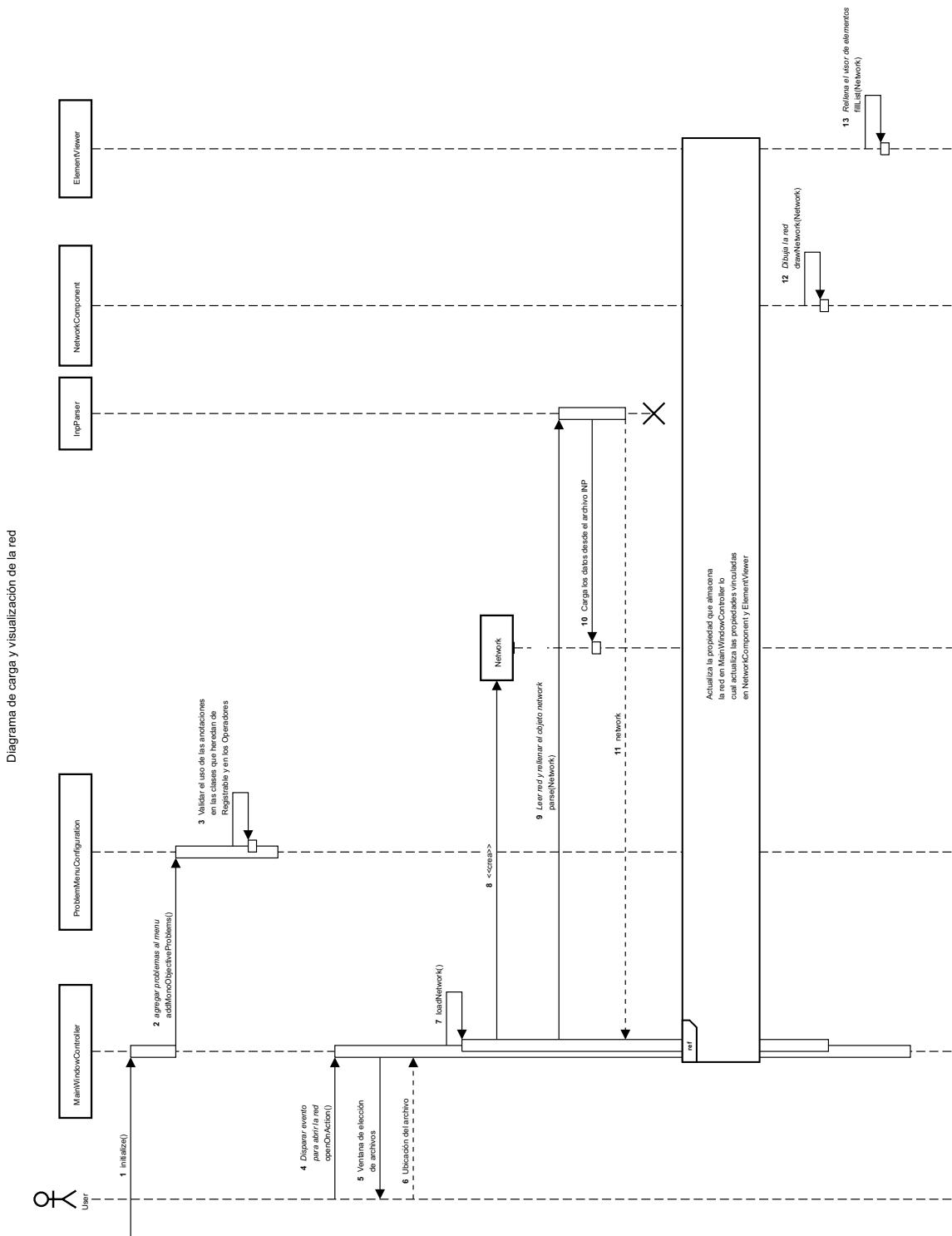


Figura 3.5: Diagrama de secuencia de la carga y visualización de la red.

Diagrama para la ejecución de la simulación hidráulica

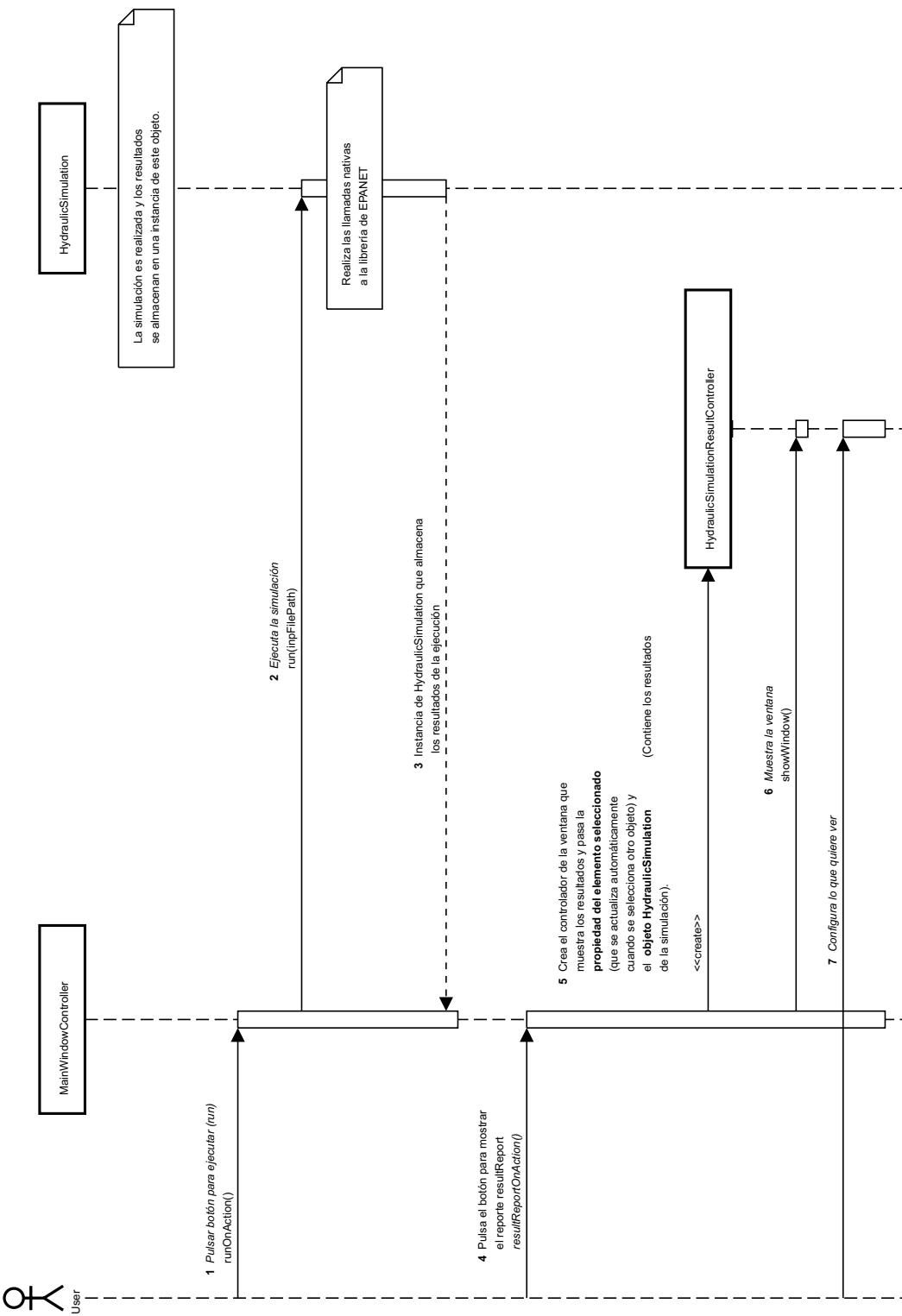


Figura 3.6: Diagrama de secuencia de la simulación de la red usando los valores del archivo de configuración de red (inp).

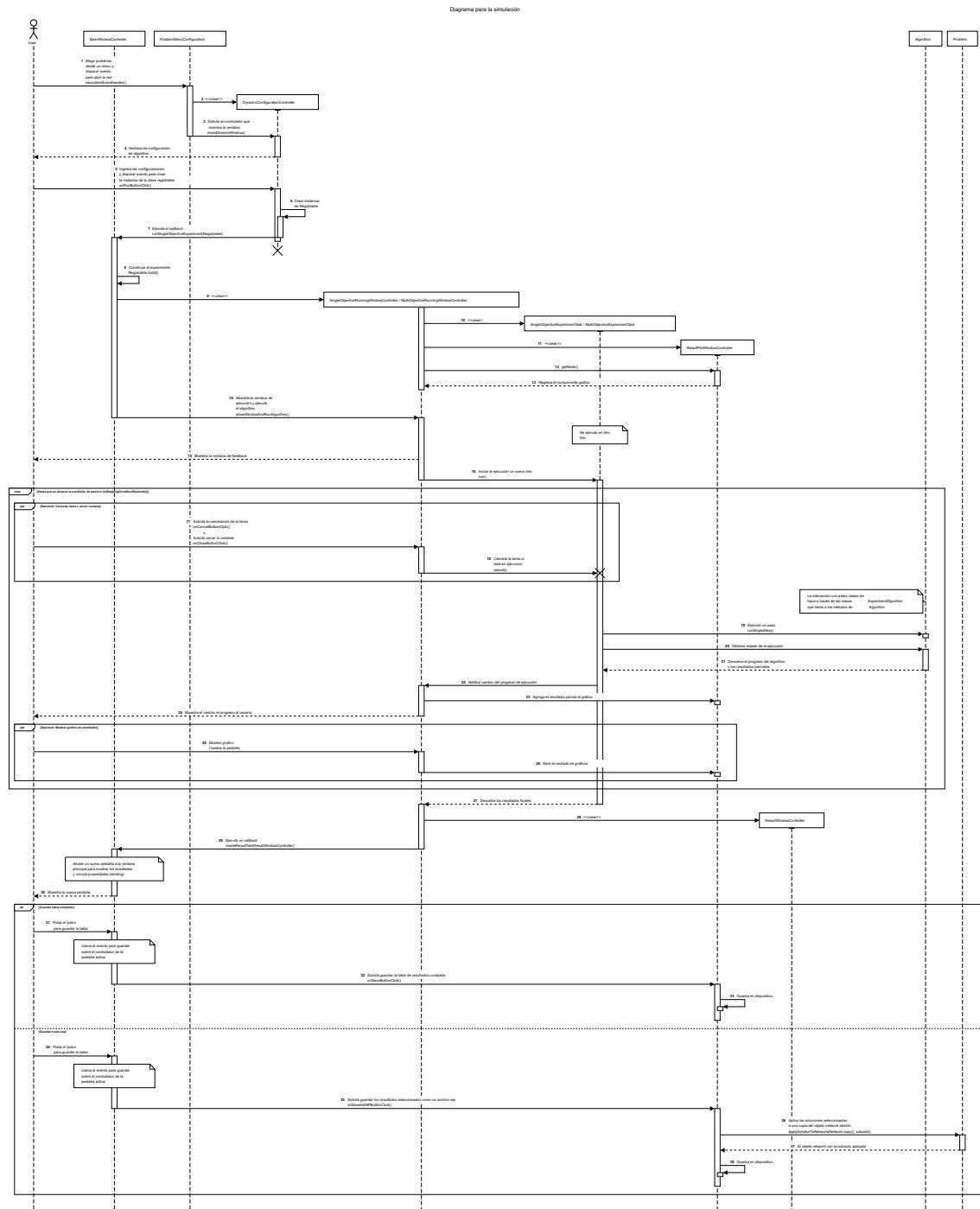


Figura 3.7: Diagrama de secuencia de la optimización.

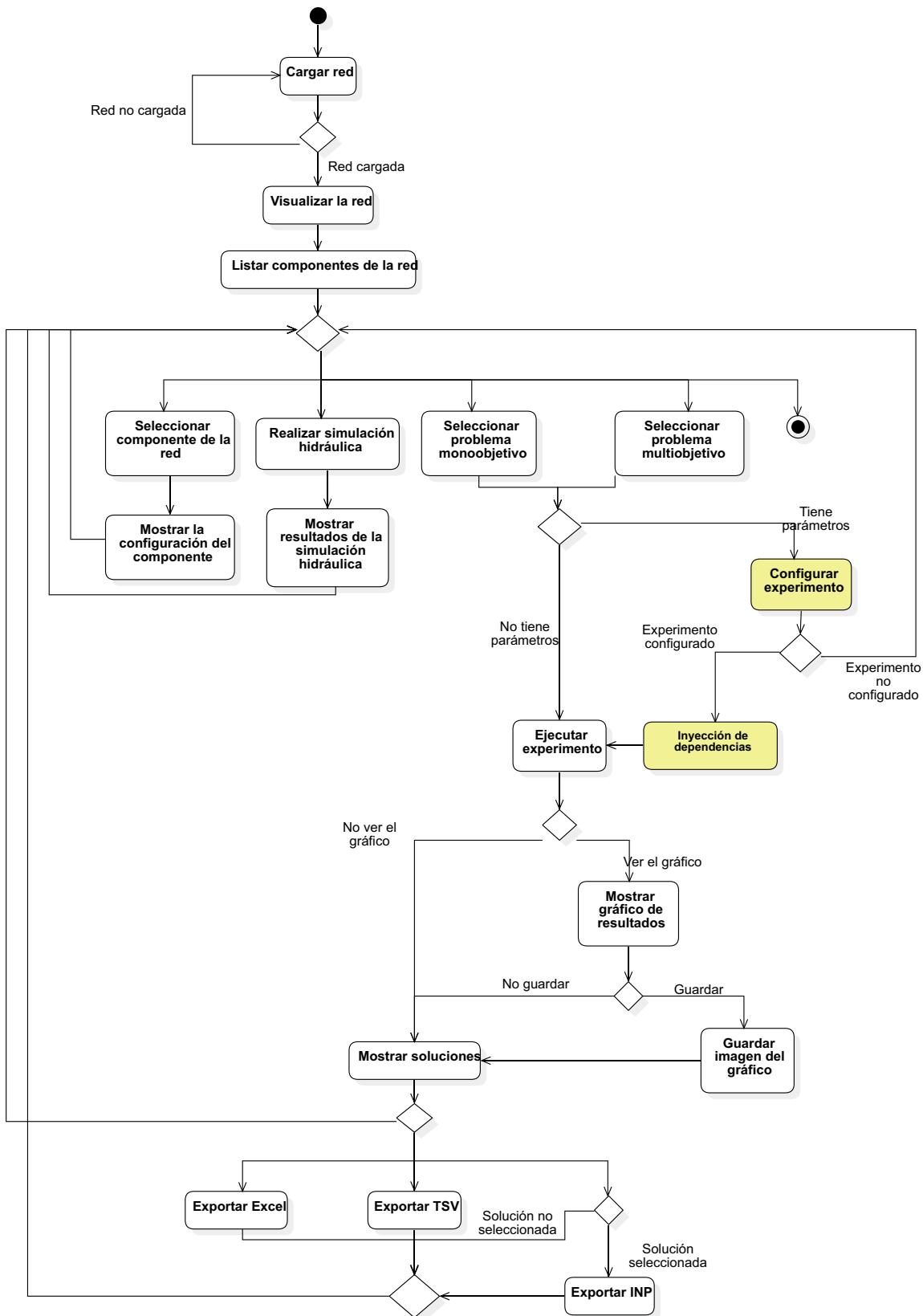


Figura 3.8: Diagrama de actividades

4. Diseño de interfaces

Durante este capítulo se presentan las interfaces que conforman la aplicación.

4.1. Interfaz de inicio

La interfaz de inicio se divide en tres secciones, el menú, el visualizador de red y un apartado para ver los elementos de la red. Para cargar una red debe ir a *File > Open*. Para resolver un problema debe ir al menú *Problems* y elegir el problema a resolver. Para ver más detalles de un componente de la red, entonces pulse dos veces en componente de la red en el apartado de elementos. Esta interfaz se muestra en la Figura 4.1.

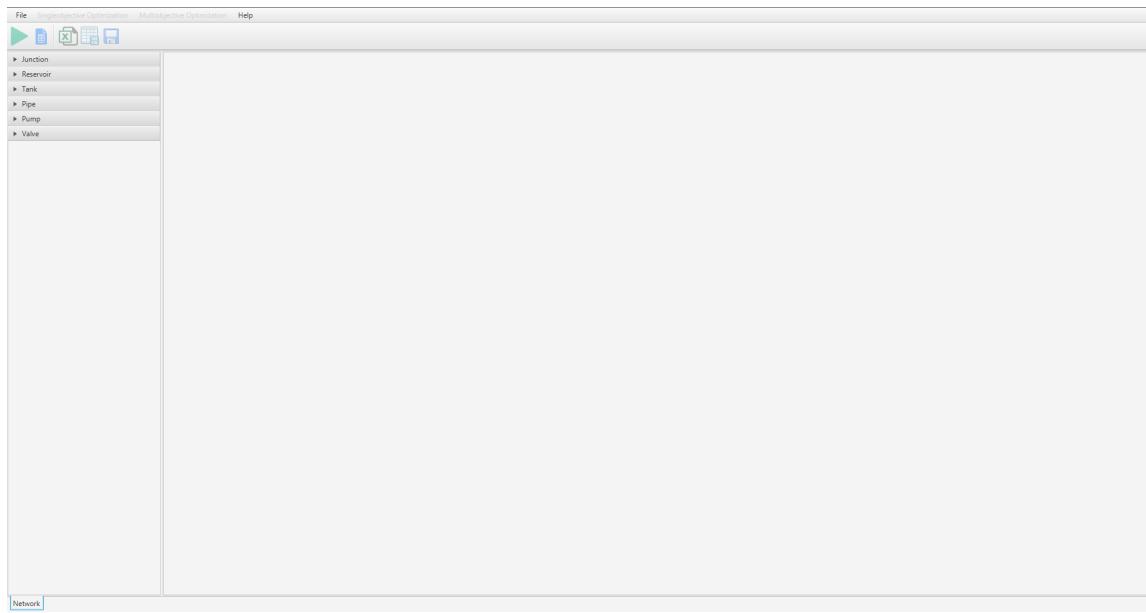


Figura 4.1: Esquema de interfaz de inicio del sistema.

4.2. Interfaz de configuración y descripción del problema

La interfaz de configuración y descripción del problema es mostrada cuando el problema tiene parámetros que configurar. La Figura 4.2 muestra la interfaz de descripción del problema a optimizar. La Figura 4.3 muestra la interfaz para configurar los parámetros del problema.

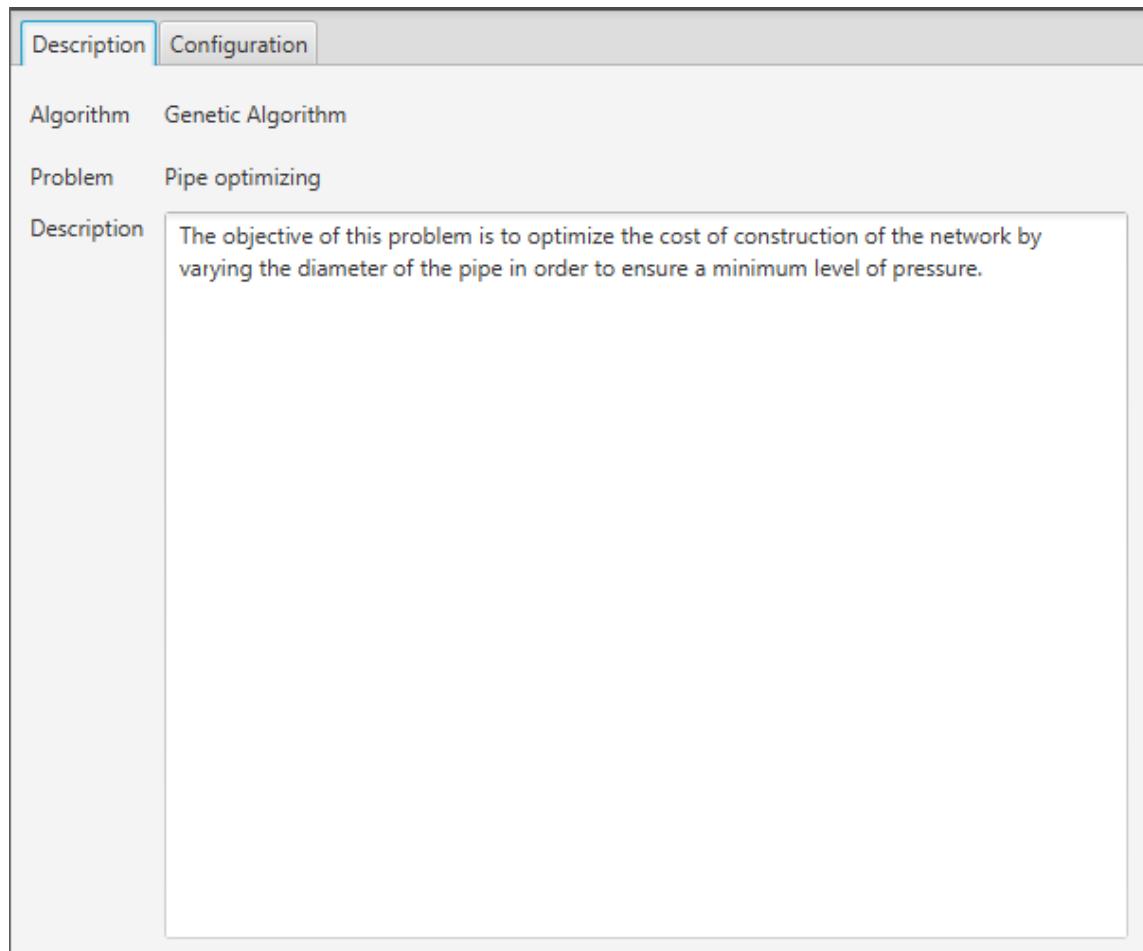


Figura 4.2: Interfaz de descripción del problema.

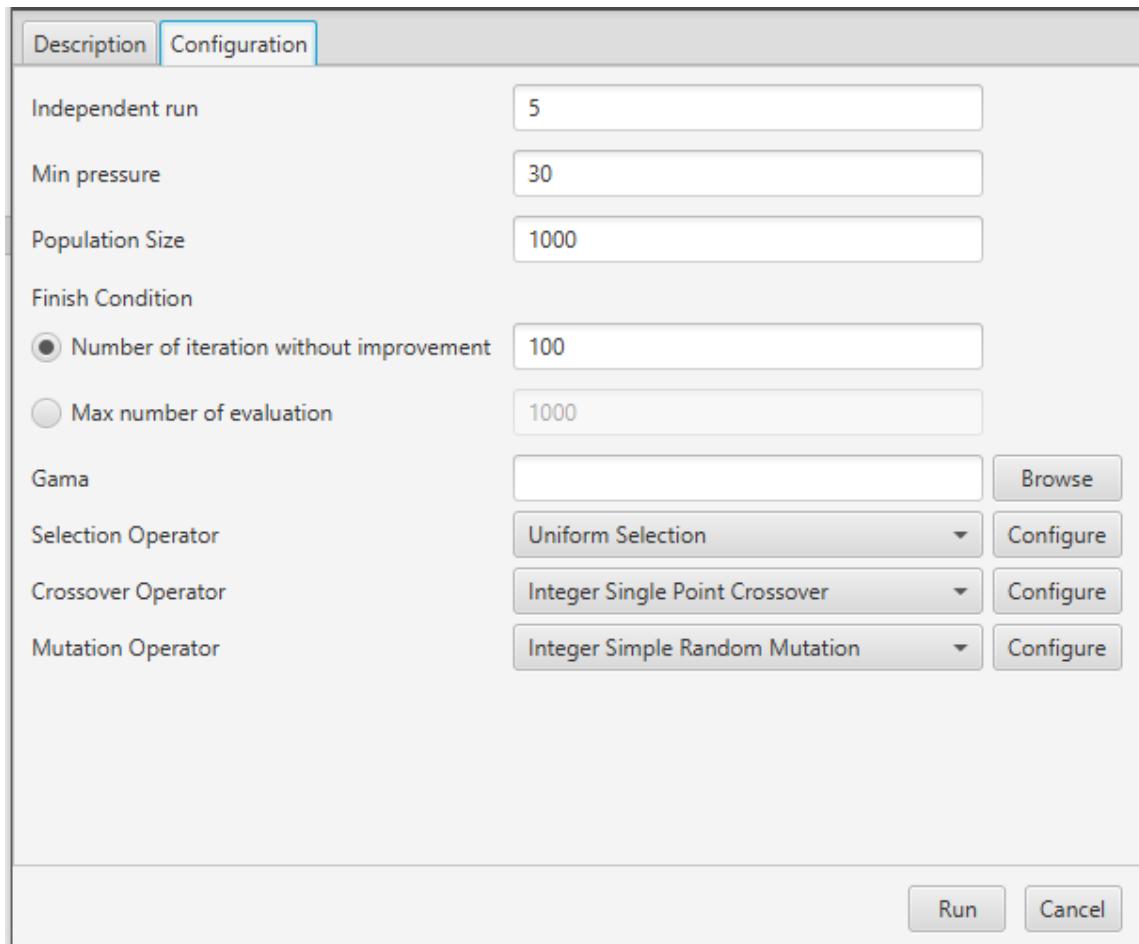


Figura 4.3: Interfaz de configuración del problema.

4.3. Interfaz de ejecución del experimento

Esta interfaz es mostrada mientras se lleva a cabo la ejecución del algoritmo. Existen 2 interfaces de ejecución. Una para el problema monoobjetivo y otra para el multiobjetivo. La Figura 4.4 muestra la interfaz usada cuando el problema es monoobjetivo. Por otro lado, la Figura 4.5 muestra la interfaz para los problemas multiobjetivos.

Si se presiona el botón cancelar, entonces la ejecución del algoritmo será detenida.

La pestaña “*Chart*” está disponible para la interfaz de los problemas multiobjetivos de hasta dos objetivos.

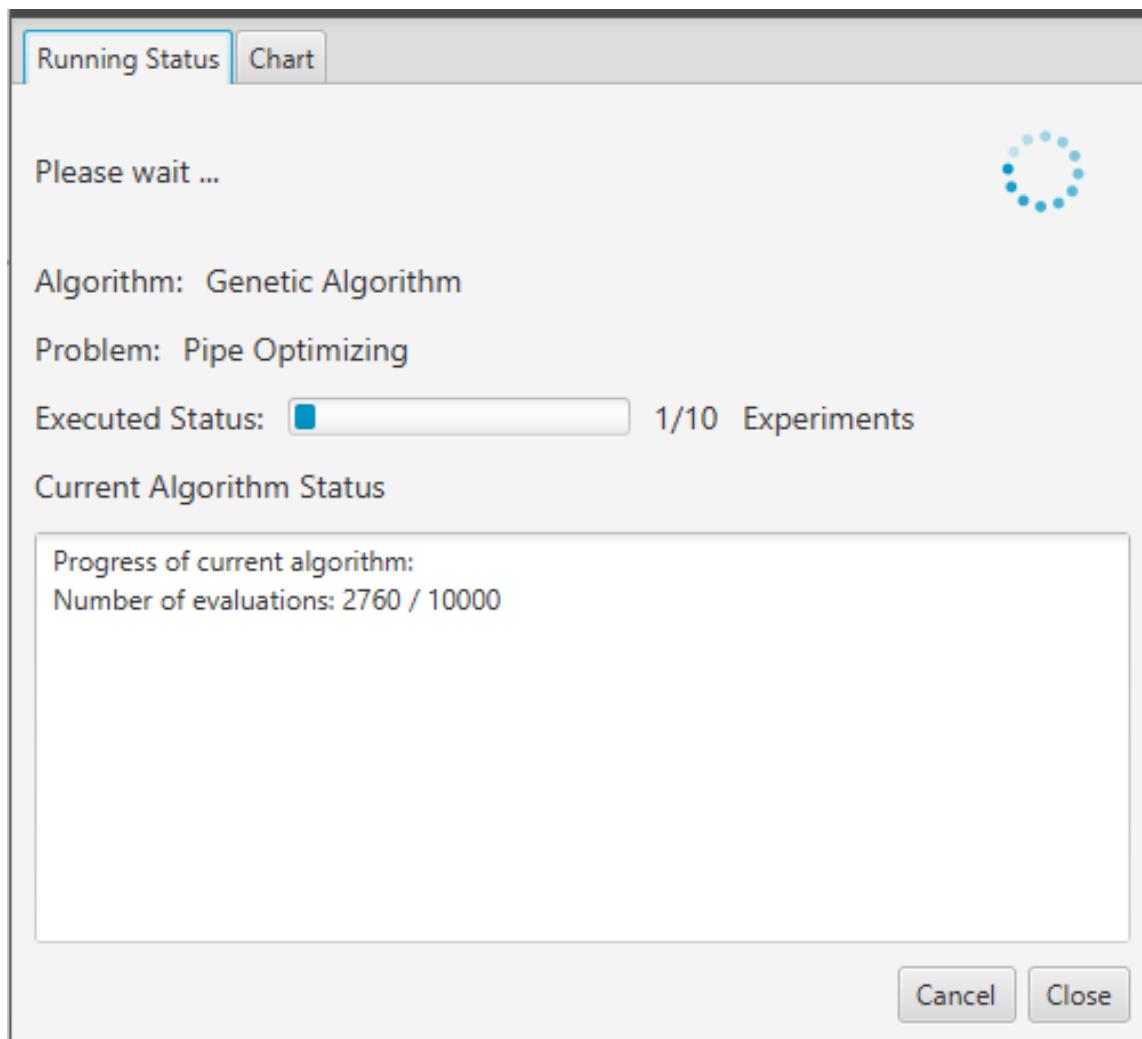


Figura 4.4: Interfaz de ejecución del experimento monoobjetivo.

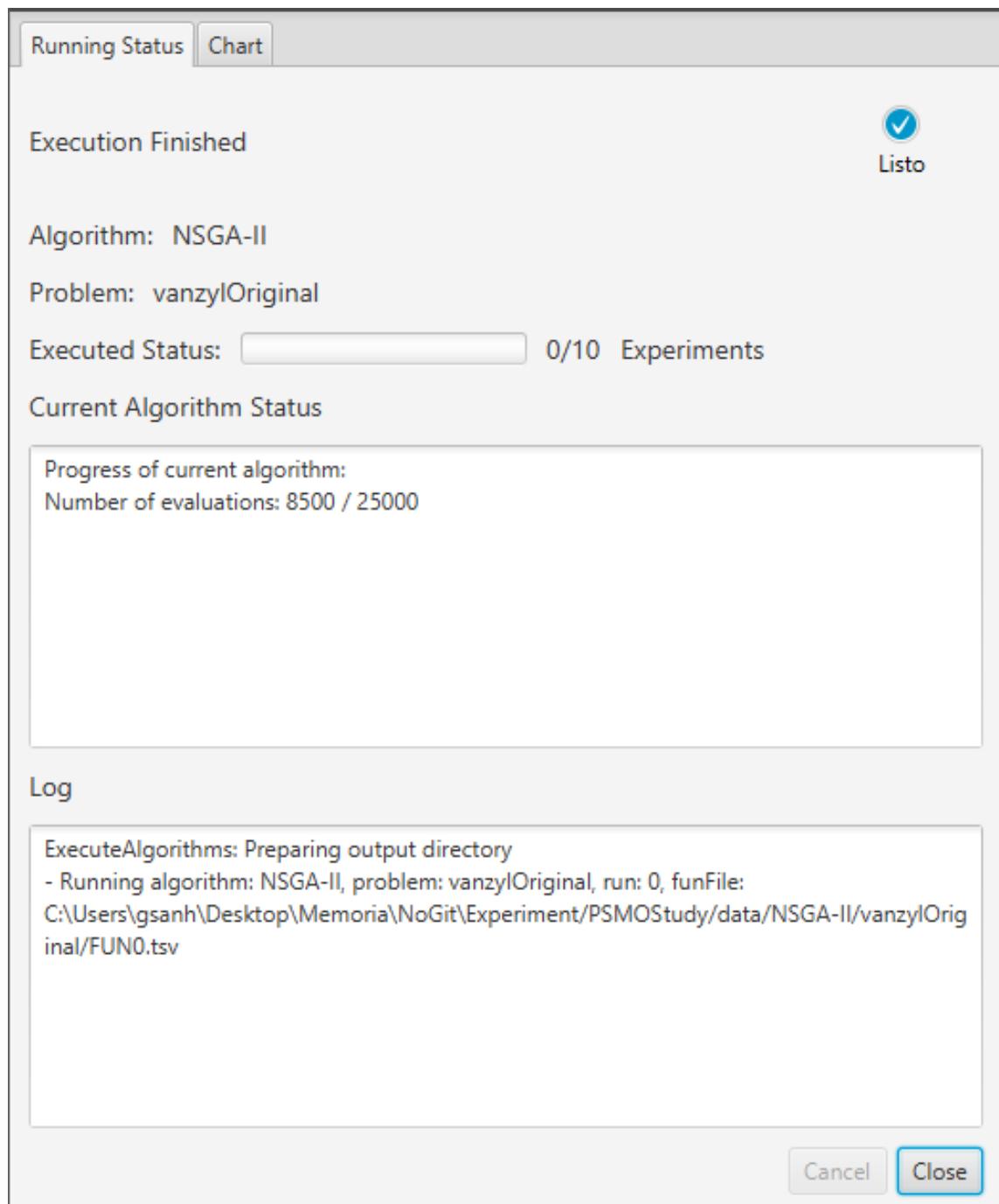


Figura 4.5: Interfaz de ejecución del experimento multiobjetivo.

4.4. Interfaz de gráficos

Esta interfaz es mostrada cuando se selecciona la pestaña “*Chart*”. Ésta cuenta con un gráfico de dos ejes. Si el problema es de un objetivo, entonces el eje vertical corresponde al valor del objetivo después de realizar la evaluación, mientras que el eje horizontal corresponde al número de generaciones. Si el problema tiene dos objetivos, el eje vertical corresponde al primer objetivo y el eje horizontal corresponde al segundo objetivo. La interfaz para problemas monoobjetivos es mostrada en la Figura 4.6, mientras que para el problema multiobjetivo se muestra la interfaz en la Figura 4.7.

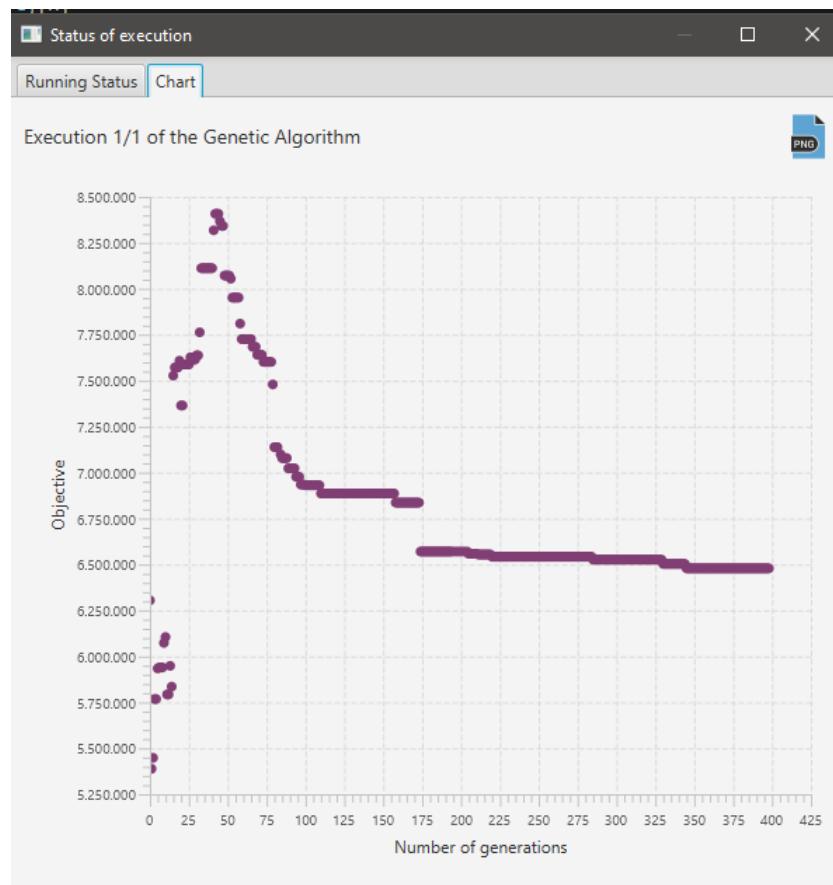


Figura 4.6: Interfaz de gráfico de soluciones para problemas monoobjetivos

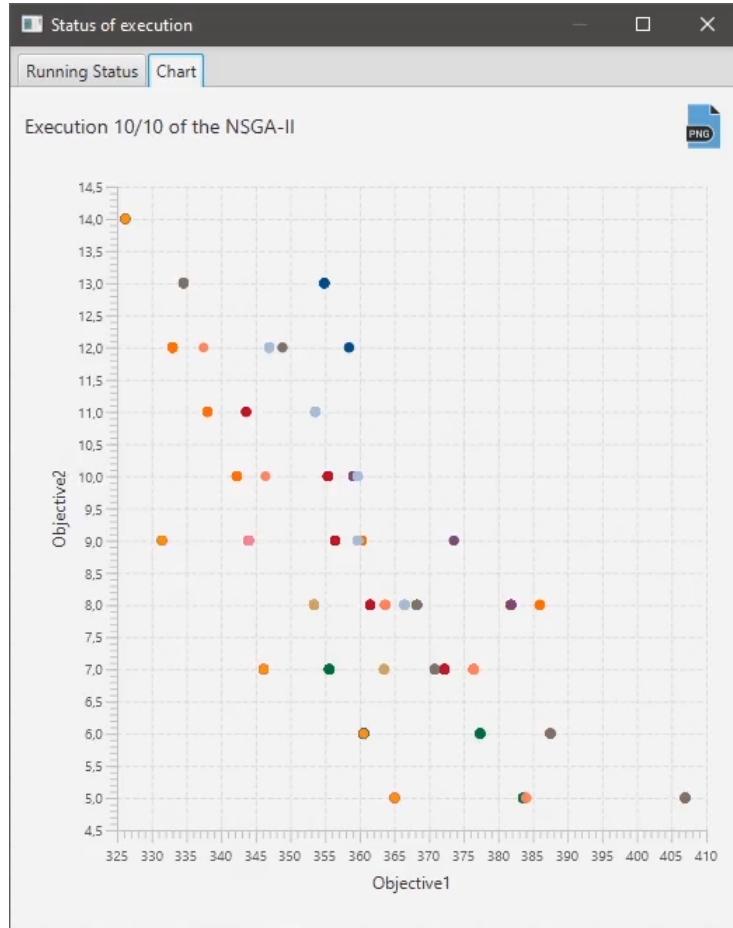


Figura 4.7: Interfaz de gráfico de soluciones para problemas multiobjetivos

4.5. Interfaz de resultados

La interfaz de resultados es mostrada cuando la ejecución del algoritmo ha terminado exitosamente. Esta interfaz es mostrada en la ventana principal de la aplicación en una nueva pestaña. Al seleccionar una pestaña de resultado se activan tres botones como se muestra en la Figura 4.8.

El botón “*Save selected ítem as inp*” crea un inp para la solución seleccionada. Para esto se envía una copia del objeto Network abierto y la solución al método applySolutionToNetwork, al objeto problema. Este método sustituye en el objeto Network los valores correspondientes indicados en la solución y devuelve nuevamente el objeto Network para poder guardarlo usando un objeto que implementa la interfaz

OutputWriter.

El botón “Save Table” guarda todas las soluciones, en dos archivos separados. Uno de estos archivos guarda solamente las variables de decisión, y el otro guarda los valores de los objetivos. El nombre de éstos corresponde al dado a través del FileChooser que es mostrado al presionar el botón. A este nombre se le agrega el sufijo -FUN, para el archivo con los valores de los objetivos; y -VAR, para el archivo con los valores de las variables de decisión.

El botón “Save Table as Excel” exporta la tabla a una planilla Excel.

Objective 1	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X25	X26	X27	X28	X29	X30	X31	X32	X33	X34	Overall Constrain Violation	Number of Generation	Min Pressure	Popula
6700749.918494301	6	6	4	4	1	3	4	5	6	6	6	5	4	1	6	3	1	4	4	5	3	6	2	1	1	2	4	6	6	5	5	6	0.0	1000	30	10		
6608527.89138204	6	5	4	5	6	4	6	6	6	6	6	3	1	6	4	3	1	1	3	2	5	2	1	1	1	4	6	5	5	3	2	1	0.0	1000	30	10		
6499676.728710202	6	4	5	4	5	5	6	6	6	6	6	3	1	6	5	5	2	1	1	2	5	2	1	1	1	4	5	4	3	3	1	1	0.0	1000	30	10		
6899153.958442001	6	5	4	5	4	3	4	5	5	6	6	6	3	1	6	4	2	3	1	1	5	6	4	3	3	4	1	6	5	6	5	5	0.0	1000	30	10		
6953245.571548505	6	5	5	5	2	2	3	6	6	5	6	5	4	2	6	3	1	3	4	5	3	6	3	2	1	1	4	6	6	5	5	4	5	0.0	1000	30	10	
6428017.77767302	6	5	4	4	5	5	6	6	6	6	6	6	3	1	6	5	3	2	2	1	3	5	3	1	1	2	3	4	5	4	2	1	1	0.0	1000	30	10	
6389710.047553201	6	5	4	4	1	1	2	4	4	5	4	6	3	1	6	4	5	2	1	1	3	6	4	3	1	1	2	6	6	5	5	5	0.0	1000	30	10		
6591050.15372009	6	5	5	4	5	5	6	6	6	6	6	6	3	2	5	4	1	5	4	6	1	6	1	3	3	4	6	5	5	4	4	3	0.0	1000	30	10		
6934641.5009749	6	6	5	3	4	6	4	4	5	6	5	6	3	1	6	6	5	5	3	1	3	6	1	1	2	3	5	6	5	5	4	5	0.0	1000	30	10		
6805819.273527002	6	5	5	4	5	4	6	5	6	6	6	3	1	6	5	4	1	1	2	2	6	1	1	1	2	3	5	6	5	5	3	3	0.0	1000	30	10		

Figura 4.8: Interfaz de resultados de la optimización

4.6. Interfaz de visualización de configuraciones de la red

Al seleccionar un componente de la red en la interfaz gráfica y hacer doble click se abrirá una interfaz que permite ver la configuración por defecto de los componentes de la red. Esta interfaz consiste en mostrar una tabla en donde la primera columna será el nombre del atributo de la red y la segunda el valor. Un ejemplo de esta interfaz para un componente de la red se puede visualizar en la Figura 4.9.

Property	Value
Junction ID	N30
X-Coordinates	2971.0
Y-Coordinates	1355.0
Description	
Tag	
Elevation	0.0
Demand	100.0
Demand Pattern	
Demand Categories	1
Emitter Coeff.	
Initial Quality	
Source Quality	

Figura 4.9: Interfaz de visualización de la configuración de la red.

4.7. Interfaz de ejecución de una red

Se puede ejecutar una red con su configuración por defecto. Una vez ejecutada la red se puede visualizar en una interfaz los resultados de su ejecución. Este interfaz se muestra en la Figura 4.10.

Result of execution

Select the type of table to create

Network nodes at

Network link at

Time series for nodes

Time series for links

OK **Cancel**

Node ID	Demand	Head	Pressure	Quality
n1	0.0	19.999849319458008	9.999848365783691	0.0
n10	0.0	19.999807357788086	-80.00019073486328	0.0
n12	0.0	19.999807357788086	-80.00019073486328	0.0
n11	0.0	109.22994232177734	9.229939460754395	0.0
n13	0.0	109.22994232177734	9.229939460754395	0.0
n2	0.0	109.22989654541016	99.22989654541016	0.0
n3	0.0	89.96748352050781	14.967482566833496	0.0
n361	0.0	89.9674301147461	-10.032569885253906	0.0
n362	0.0	89.96737670898438	-10.032622337341309	0.0
n364	0.0	111.8226089477539	11.822610855102539	0.0
n365	0.0	111.82255554199219	11.822559356689453	0.0
n5	31.0	84.49990844726562	54.499908447265625	0.0
n6	62.0	84.50836944580078	54.50837326049805	0.0
r1	-241.30252075195312	20.0	0.0	0.0
t6	42.83230972290039	94.5	9.5	0.0
t5	105.47020721435547	84.5	4.5	0.0

Figura 4.10: Interfaz de resultados de ejecución de simulación hidráulica.

Los botones “Time series for nodes” y “Time series for link” solo son mostrados cuando la red está configurada para simular más de un periodo de tiempo de simulación.

5. Detalles de implementación

Una necesidad del sistema es poder añadir nuevos algoritmos, operadores y problemas. Sin embargo, para hacer uso de éstos desde la interfaz de usuario, es necesario que el implementador modifique manualmente la interfaz. Ésto lleva una carga extra al implementador al tener que aprender la tecnología necesaria que fue usada para crear la interfaz, la cual para este sistema corresponde a JavaFX. Debido a estas razones, para facilitar el trabajo del implementador se tomó como alternativa usar dos funcionalidades del lenguaje de Java, las cuales son *Java Reflection* y *Java Annotation*. El uso que se hace por parte de ellas en el sistema es:

Java Reflection Examina las clases durante la ejecución del programa con el fin de construir una interfaz de usuario que permita configurar los valores que son necesarios al momento de crear un objeto de dicha clase.

Java Annotation Agrega metadatos a los elementos del programa, en este caso a los constructores, que son leídos a través de la *Java Reflection API*. Estos metadatos contienen información del constructor como el nombre de los parámetros que recibe y en el caso de que el parámetro sea un objeto, las alternativas de las clases para crear dicho objeto.

Haciendo uso de estas dos tecnologías, se puede reducir el problema anteriormente mencionado, puesto que, estableciendo y siguiendo una convención se puede crear dinámicamente una la *GUI* para instanciar nuevos objetos sin conocer su tipo previamente en tiempo de compilación. La convención anteriormente mencionada, consiste en implementar una interfaz, en la cual el constructor indique los parámetros que requiere. Estos parámetros deben estar escritos en un orden determinado,

para crear y configurar el algoritmo cuando éste sea solicitado. El nombre de dicha interfaz es *Registrable*.

5.1. Uso de *Java Annotation* y *Java Reflection*

Las anotaciones presentes en el sistema, su finalidad y donde deben ser usadas se define a continuación:

5.1.1. Anotaciones para los operadores

@DefaultConstructor

Indica el constructor que debe ser usado al momento de crear una instancia del operador. Esta anotación recibe un arreglo de *NumberInput*, el cual se define en la siguiente sección. El arreglo debe tener la misma cantidad de argumentos que los parámetros del constructor como se muestra en la Figura 5.1 y Figura 5.2.

```
@DefaultConstructor(@NumberInput(displayName = "constant", defaultValue = 1.5))
public UniformSelection(double constant)
```

Figura 5.1: Constructor de un solo parámetro.

```
@DefaultConstructor({
    @NumberInput(displayName = "MutationProbability", defaultValue = 0.1),
    @NumberInput(displayName = "DistributionIndex", defaultValue = 0.1
})
public IntegerPolynomialMutation(double mutationProbability, double distributionIndex)
```

Figura 5.2: Constructor de un solo parámetro.

Esta anotación solo puede ser usada en un único constructor por clase. Usar esta anotación en más de un constructor lanzara una excepción en tiempo de ejecución. Adicionalmente, el constructor que use esta anotación solo puede tener parámetros de tipo *int* o *double*.

La interfaz gráfica creada para cada anotación se puede ver en la Figura 5.3 y 5.4.



Figura 5.3: Interfaz para configurar el operador *UniformSelection*.

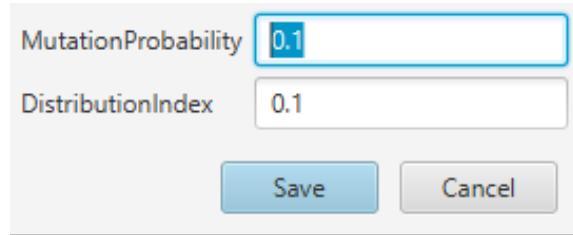


Figura 5.4: Interfaz para configurar el operador *IntegerPolynomialMutation*.

La anotación puede tener un arreglo vacío, lo cual indica que el constructor no recibirá parámetros.

5.1.2. Anotaciones para los objetos que heredan la interfaz *Registrable* *@NewProblem*

Esta anotación permite indicar el nombre del problema que es mostrado en la interfaz gráfica. Puedes ver el uso de esta anotación en la Figura 5.5.

```

@NewProblem(displayName = "Pipe optimizing", algorithmName = "Genetic Algorithm",
    description = "The objective of this problem is to optimize the cost of "
        "construction of the network by varying the diameter of the pipe in order "
        "to ensure a minimum level of pressure.")
@Parameters(operators = {
    @OperatorInput(displayName = "Selection Operator", value = {
        @OperatorOption(displayName = "Uniform Selection", value = UniformSelection.class)
    }),
    @OperatorInput(displayName = "Crossover Operator", value = {
        @OperatorOption(displayName = "Integer Single Point Crossover", value = IntegerSinglePointCrossover.class),
        @OperatorOption(displayName = "Integer SBX Crossover", value = IntegerSBXCrossover.class)
    }),
    @OperatorInput(displayName = "Mutation Operator", value = {
        @OperatorOption(displayName = "Integer Simple Random Mutation", value = IntegerSimpleRandomMutation.class),
        @OperatorOption(displayName = "Integer Polynomial Mutation", value = IntegerPolynomialMutation.class),
        @OperatorOption(displayName = "Integer Range Random Mutation", value = IntegerRangeRandomMutation.class)
    })
}, // files = {
    @FileInput(displayName = "Gama")
}, // numbers = {
    @NumberInput(displayName = "Independent run", defaultValue = 5),
    @NumberInput(displayName = "Min pressure", defaultValue = 30),
    @NumberInput(displayName = "Population Size", defaultValue = 1000)
}, // numbersToggle = {
    @NumberToggleInput(groupID = "Finish Condition", displayName = "Number of iteration without improvement", defaultValue = 100),
    @NumberToggleInput(groupID = "Finish Condition", displayName = "Max number of evaluation", defaultValue = 10000)
})
public PipeOptimizingRegister(Object selectionOperator, Object crossoverOperator, Object mutationOperator, File gama, int independentRun,
    int minPressure, int populationSize, int maxEvaluations) throws Exception

```

Figura 5.5: Constructor de clase que hereda de registrable y sus metadatos para cada parámetro.

Los elementos en esta anotación consisten en:

- *displayName*: El nombre del problema. Este nombre también actúa como el nombre de la categoría.
- *algorithm*: Un *String* con el nombre del algoritmo usado para resolver el problema.
- *description*: Un *String* con la descripción del algoritmo.

El nombre dado en el elemento *displayName*, es el nombre visible del problema en el menú de la aplicación y permite agrupar a los problemas que tengan el mismo nombre, pero distintos algoritmos, como se ve en la Figura 5.6.

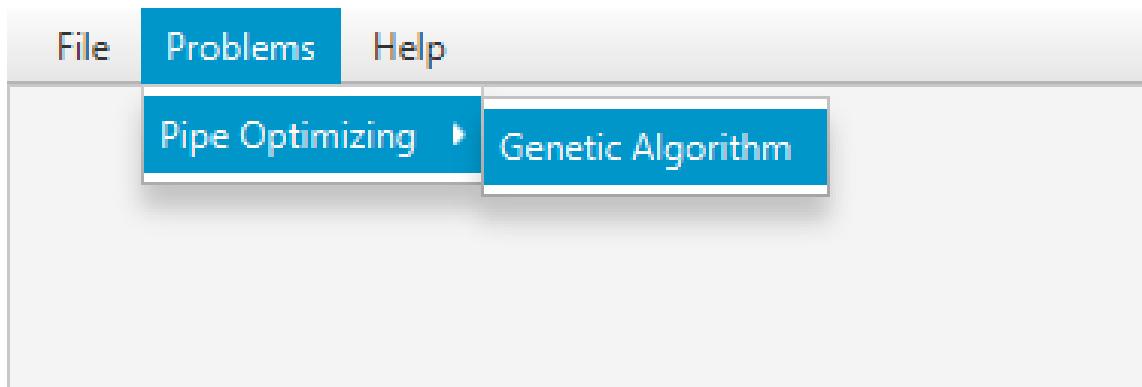


Figura 5.6: Menú de problemas.

Esta anotación solo puede estar en un constructor, en caso de esta anotación no esté presente, se lanza un error en tiempo de ejecución.

@Parameters

Esta anotación permite agregar información acerca de los parámetros recibidos por el constructor. Cuando el constructor tiene esta anotación, por convención, está obligado a declarar los parámetros en un orden determinado en base a tu tipo. Este orden es el siguiente:

1. *Object*

2. *File*
3. *int o double*

Si el constructor no declara los parámetros en ese orden un error en tiempo de ejecución es lanzado.

Dentro de esta anotación existen varios elementos. Estos elementos son:

- *operators*: Arreglo que recibe anotaciones del tipo *OperatorInput*.
- *files*: Arreglo que recibe anotaciones del tipo *FileInput*.
- *numbers*: Arreglo que recibe anotaciones del tipo *NumberInput*.
- *numbersToggle*: Arreglo que recibe anotaciones del tipo *NumberToggleInput*.

El valor por defecto para todos los elementos mencionados anteriormente es un arreglo vacío ({}). No usar la anotación *@Parameters* tiene el mismo efecto que usar la anotación, pero con sus valores por defecto.

Puede ver el uso de esta anotación en la Figura 5.5.

@OperatorInput

Esta anotación agrega información a uno de los parámetros del constructor acerca de los posibles operadores que pueden ser recibidos por ese parámetro. Dentro de esta anotación existen varios elementos. Estos elementos son:

1. *displayName*: Nombre de categoría para los operadores.
2. *value*: Arreglo que recibe anotaciones del tipo *OperatorOption*.

En la ventana de configuración de problema, estos parámetros son vistos con un *ComboBox* como muestra la Figura 5.7.

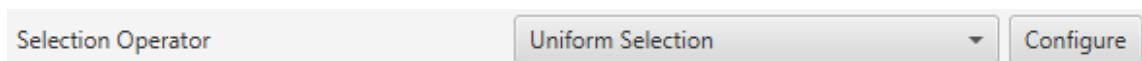


Figura 5.7: Componente *ComboBox* para configurar los operadores.

Las alternativas disponibles dentro del *ComboBox* están dadas por aquellas indicadas en el elemento *value* de este operador. Como se muestra en la Figura 5.5, la única alternativa para el *Selection Operator* es el operador *UniformSelection* apreciado en la Figura 5.8.



Figura 5.8: *ComboBox* expandido para configurar el operador.

Por defecto, el *ComboBox* selecciona el primer elemento de la lista. El botón *Configure* permite configurar los parámetros que recibe el constructor del operador, aquel que posee la anotación *@DefaultConstructor*. Para el caso del operador *UniformSelection*, su interfaz de configuración se muestra en la 5.3.

@OperatorOption

Esta anotación permite indicar las alternativas de operadores que puede recibir un parámetro para una categoría de operador indicada por la anotación *@OperatorInput*. Dentro de esta anotación existen varios elementos. Estos elementos son:

- *displayName*: Nombre del operador. Este es el nombre visualizado en el *ComboBox* como se muestra en la Figura 5.8.
- *value*: Instancia del tipo *Class* que referencia el tipo de operador.

@FileInput

Esta anotación indica que hay un parámetro que espera recibir un objeto de tipo *File*. Cuando esta anotación está presente junto con su parámetro, en la interfaz, aparecerá un apartado que abre un *FileChooser* o un *DirectoryChooser* para buscar un archivo o directorio, respectivamente. Dentro de esta anotación existen solo un elemento. Éste es:

- *displayName*: Nombre del parámetro. Este nombre también corresponde al nombre visualizado en la ventana de configuración como muestra la Figura 5.9.

- *type*: Indica el modo en que se abrirá el *FileChooser*. Este elemento recibe un enumerado del tipo *Type*; los cuales son *Type.OPEN*, *Type.SAVE*, que abren un *FileChooser* para leer o guardar un archivo; y *Type.Directory*, el cual abre un *DirectoryChooser* para seleccionar un directorio. La opción por defecto es *FileType.OPEN*.

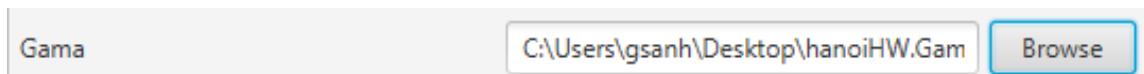


Figura 5.9: Apartado para configurar el parámetro de tipo *File*.

Si el *TextField* donde se muestra la ruta está vacío, es decir, no se ha seleccionado un archivo o carpeta, entonces será inyectado ***null*** en el parámetro correspondiente del constructor.

@NumberInput

Esta anotación indica que hay un parámetro del tipo *int* o *double* o sus tipos envoltorios *Integer* o *Double*, respectivamente. Esta anotación agrega en la interfaz un *TextField* que solo permite como entrada un número. Si el tipo del parámetro es *int* o *Integer*, entonces el *TextField* solo permite ingresar números enteros. Por otro lado, si el parámetro es *double* o *Double*, entonces en la interfaz se acepta el ingreso de números enteros o decimales. El apartado para esta anotación se muestra en la Figura 5.10.

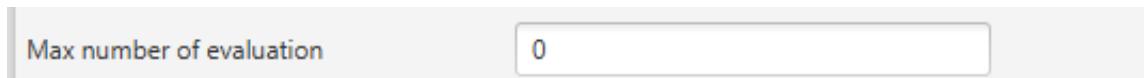


Figura 5.10: *TextField* presente cuando esta la anotación **@NumberInput**.

Dentro de esta anotación existen los siguientes elementos:

- *displayName*: Nombre del parámetro.
- *defaultValue*: Valor por defecto de la propiedad. Si el tipo de parámetro en el constructor de la clase que hereda de *Registrable* es un entero, pero se ingresa

un valor con decimales, los decimales serán truncados. Si este elemento no se define su valor por defecto es 0.

@NumberToggleInput

Esta anotación indica que hay un conjunto de parámetros que son mutuamente excluyentes entre ellos, es decir, que solo un parámetro puede recibir el valor. En la interfaz, el nombre del grupo aparece sobre los componentes. Dentro de un mismo grupo solo se puede configurar un parámetro. El parámetro por configurar debe ser indicado activando el *ToggleButton* correspondiente, lo cual conlleva a la activación del *TextField*. Dentro de esta anotación existen varios elementos. Estos elementos son:

- *groupID*: *String* con un id para el grupo. Las anotaciones *NumberToggleInput* que tengan el mismo id, en la interfaz, se encuentran en una sección cuyo título es el nombre del grupo. Esto se aprecia en la Figura 5.11.
- *displayName*: Nombre del parámetro.
- *defaultValue*: Valor por defecto de la propiedad. Si el tipo de parámetro en el constructor de la clase que hereda de *Registrable* es un entero, pero se ingresa un valor con decimales, los decimales serán truncados. Si este elemento no se define su valor por defecto es 0.

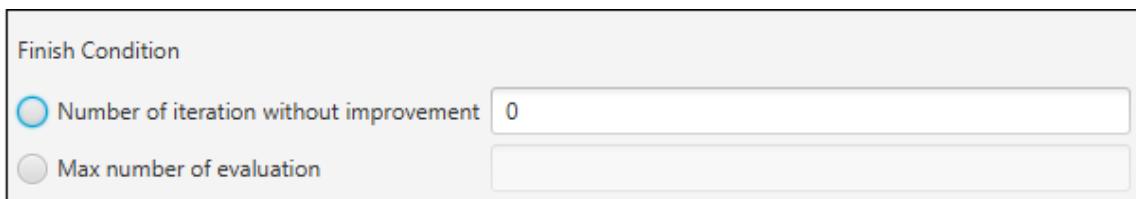


Figura 5.11: Apartado para *NumberToggleInput* con el mismo GroupID.

El parámetro configurado en la interfaz de usuario recibe el valor indicado en el *TextField*. Si el *TextField* queda vacío entonces recibe el valor cero. Sin embargo, los demás parámetros, cuyos *TextField* están deshabilitados, van a recibir el

valor ***Double.MIN_VALUE***, si el parámetro es de tipo *double* o *Double*; o ***Integer.MIN_VALUE*** si el parámetro es de tipo *int* o *Integer*. Por ejemplo, en la Figura 5.11, se observa que el parámetro “*Number of iteration without improvement*” esta activado, pero no contiene un valor, entonces al crear la instancia el constructor va a recibir el valor cero. Pero el parámetro “*Max number of evaluation*”, al no haber sido escogido, recibe el valor *Integer.MIN_VALUE*, puesto que este parámetro era de tipo *int* o *Integer*.

En la Figura 5.5 se puede ver el uso de la anotación *NumberToggleInput*. El componente que representa esta anotación en la GUI se puede apreciar en la Figura 5.11. El *groupID* es usado para nombrar la sección.

En el elemento *numbersToggle* de la anotación *@Parameters*, las anotaciones que pertenezcan al mismo grupo deben estar continuas. En caso de que esto no se cumpla se lanza una excepción al momento de ejecutar la aplicación.

Las anotaciones presentadas en las dos secciones anteriores deben ser usadas en constructores públicos.

5.2. Interfaz Registrable

Esta interfaz declara el método *build* y *getParameters*. La declaración del método *build* corresponde a la siguiente:

```
R build(String inpPath) throws Exception;
```

Donde R corresponde al tipo de valor devuelto por la función.

De esta clase se heredan dos subinterfaces. La primera corresponde a *SingleObjectiveRegistrable* que debe ser usada para implementar los problemas monoobjetivos. En cuanto a la segunda, esta corresponde a *MultiObjectiveRegistrable*, la cual se debe usar para los problemas multiobjetivos. El método *build* sobreescrito por estas clases tiene la siguiente firma:

```
Experiment<?> build(String inpPath) throws Exception;
```

donde *Experiment* consiste en una clase que almacena una lista de algoritmos (Del mismo tipo) y el problema a resolver por estos algoritmos.

Las nuevas clases deben implementar la interfaz *SingleObjectiveRegistrable* o *MultiObjectiveRegistrable*, dependiendo del tipo de problema a tratar. Las clases que implementen cualquiera de estas dos interfaces deben ser guardados en una estructura

de datos, la cual es recorrida cuando se inicia la ejecución del programa y analizada usando la *Java Reflection API*. Este análisis consiste en escanear y validar el cumplimiento de la convención establecida para las clases que implementan esta interfaz. Esta convención consiste en lo siguiente:

- La clase debe contener un único constructor que use la anotación `@NewProblem`.
- Si el constructor requiere parámetros éstos deben estar descritos usando la anotación `@Parameters`.
- El constructor debe declarar los parámetros en el siguiente orden, de acuerdo con su tipo.
 1. Object: Usado para injectar los operadores. Éstos pueden posteriormente ser casteados a su tipo correcto. La anotación correspondiente es `@OperatorInput`
 2. File: Usados cuando el problema requiere información adicional que se encuentra en un archivo diferente. La anotación correspondiente es `@FileInput`
 3. int, Integer, double o Double: Usado generalmente para configurar valores en el algoritmo o si el problema requiere otros valores que no fueron solicitados al crear los operadores. Las anotaciones correspondientes son `@NumberInput` y `@NumberToggleInput`.
 4. El constructor debe solicitar la misma cantidad de parámetros que las descritas en la anotación `@Parameters`.

Si estas convenciones no se cumplen, entonces un error en tiempo de compilación es emitido como se mencionó anteriormente en la sección anterior.

El orden en el que son injectados los parámetros consiste en el siguiente:

1. Parámetros descritos por `@OperatorInput`
2. Parámetros descritos por `@FileInput`
3. Parámetros descritos por `@NumberInput`

4. Parámetros descritos por *@NumberToggleInput*

Una vez que se haya configurado el problema a través de la interfaz se crea la instancia de la clase que hereda de Registrable y se llama a su método build, para crear el experimento y comenzar su ejecución.

La estructura de datos para registrar las clases que heredan de *SingleObjectiveRegistrable* y *MultiObjectiveRegistrable* se encuentra en la clase *RegistrableConfiguration*.

C. Manual de usuario



UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Manual de usuario

JHAWANET FRAMEWORK

TABLA DE CONTENIDOS

	página
Tabla de Contenidos	
Índice de Figuras	I
1. Como agregar un nuevo algoritmo	2
2. Como agregar un nuevo problema	4
3. Como agregar un nuevo operador	6
3.1. <i>@DefaultConstructor</i>	6
4. Como agregar un nuevo experimento	7
4.1. Interfaz Registrable	9
4.2. <i>@NewProblem</i>	13
4.3. <i>@Parameters</i>	15
4.4. <i>@OperatorInput</i>	16
4.5. <i>@OperatorOption</i>	17
4.6. <i>@FileInput</i>	17
4.7. <i>@NumberInput</i>	18
4.8. <i>@NumberToggleInput</i>	19
Glosario	21

ÍNDICE DE FIGURAS

	página
1. Diagrama de clases del módulo de metaheurísticas.	2
2. Mensaje returnedo por el método <i>getStatusOfExecution</i> en la ventana monoobjetivo.	3
3. Mensaje returnedo por el método <i>getStatusOfExecution</i> en la ventana multiobjetivo.	3
4. Método <i>runASingleStep</i> para los algoritmos evolutivos.	4
5. Ejemplo del método <i>applySolutionToNetwork</i>	5
6. Constructor de un solo parámetro.	6
7. Constructor con varios parámetros.	6
8. Interfaz para configurar el operador <i>UniformSelection</i>	7
9. Interfaz para configurar el operador <i>IntegerPolynomialMutation</i>	7
10. Diagrama módulo metaheurística extendido.	8
11. Jerarquía de clases de la interfaz Registrable.	9
12. Constructor de la clase <i>PipeOptimizingRegister</i>	11
13. Implementación del método build de la clase <i>PipeOptimizingRegister</i>	12
14. Implementación del método opcional getParameters de la clase <i>PipeOptimizingRegister</i>	13
15. Constructor de clase que hereda de registrable y sus metadatos para cada parámetro.	14
16. Menú de problemas.	15
17. Componente <i>ComboBox</i> para configurar los operadores.	16
18. <i>ComboBox</i> expandido para configurar el operador.	17
19. Apartado para configurar el parámetro de tipo <i>File</i>	18
20. <i>TextField</i> presente cuando está la anotación <i>@NumberInput</i>	18
21. Apartado para <i>NumberToggleInput</i> con el mismo GroupID.	19

1. Como agregar un nuevo algoritmo

Para agregar un nuevo algoritmo a la aplicación se debe implementar la interfaz *Algorithm*. En la Figura 1 se puede ver la interfaz correspondiente.

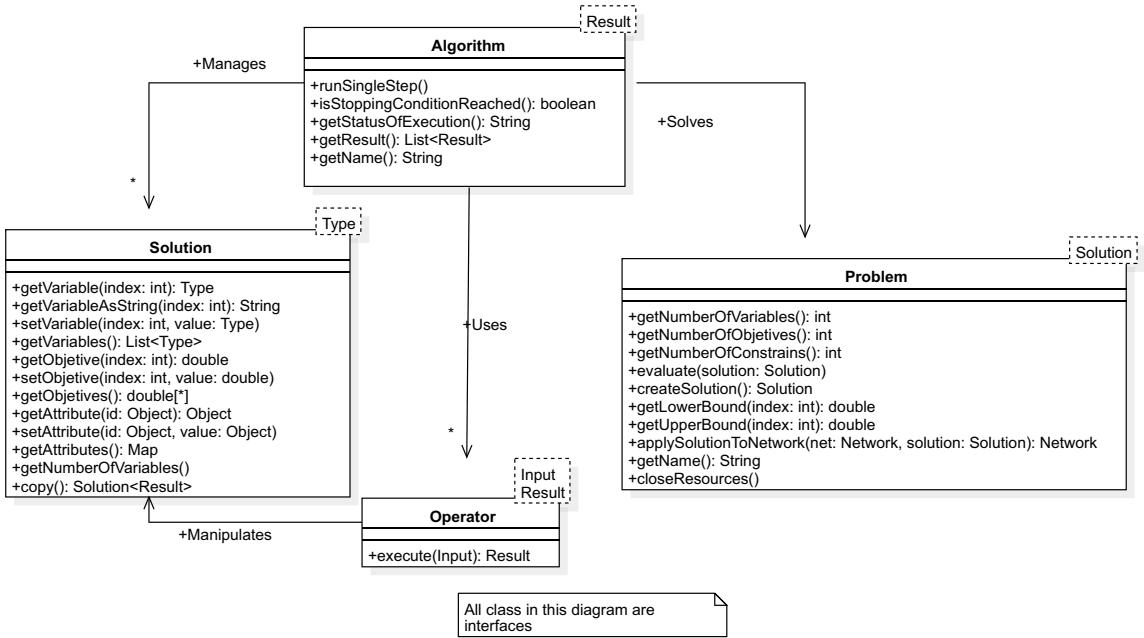


Figura 1: Diagrama de clases del módulo de metaheurísticas.

Esta interfaz cuenta con los siguientes métodos:

- *RunSingleStep*: Este método debe ejecutar un único paso del algoritmo (Una sola generación/iteración).
- *isStoppingConditionReached*: Este método indica si la condición de término del algoritmo ha sido alcanzada.
- *getStatusOfExecution*: Este método puede devolver un *String* con información del algoritmo como se muestra en la Figura 2 y 3.
- *getResult*: Devuelve el resultado del algoritmo.
- *getName*: Devuelve el nombre del algoritmo.

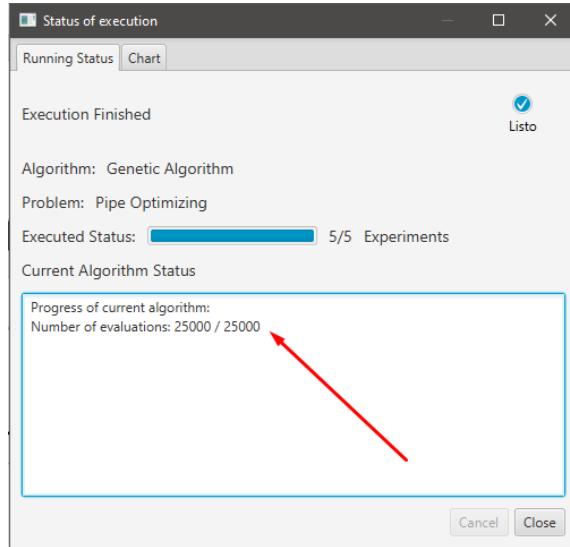


Figura 2: Mensaje returnedo por el método *getStatusOfExecution* en la ventana monoobjetivo (Está indicado por la flecha roja).

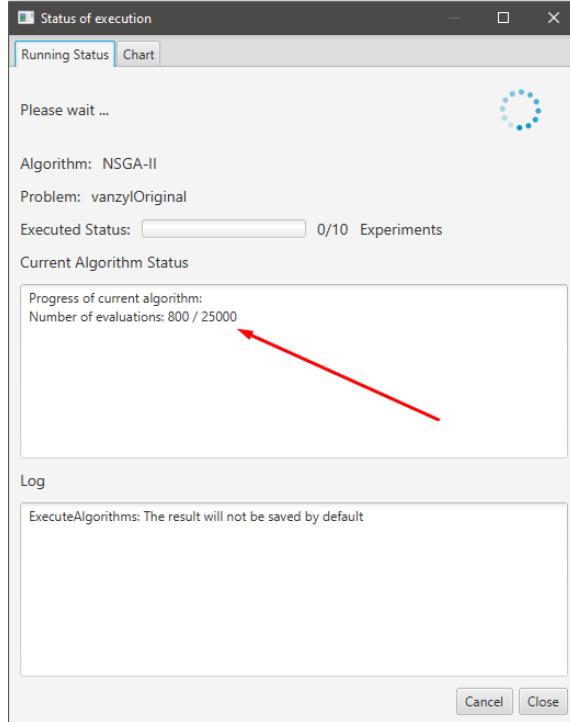


Figura 3: Mensaje returnedo por el método *getStatusOfExecution* en la ventana multiobjetivo (Está indicado por la flecha roja).

Para llevar a cabo la simulación, la aplicación llama al método *RunSingleStep* hasta que *isStoppingConditionReached* sea verdadero. Adicionalmente, dentro del mismo método *RunSingleStep* también se debería asegurar que una vez sea alcanzada la condición de termino no se realicen nuevas operaciones. Un ejemplo de *RunSingleStep* utilizado en el Algoritmo NSGAII y GA se puede ver en la Figura 4.

```

int step = 0;
List<IntegerSolution> population;
public void runSingleStep() throws Exception, EpanetException {
    List<S> offspringPopulation;
    List<S> selectionPopulation;
    // durante la primera iteración inicializa la población
    if (step == 0) {
        population = createInitialPopulation();
        population = evaluatePopulation(population);
        initProgress();
    }

    if (!isStoppingConditionReached()) {
        selectionPopulation = selection(population);
        offspringPopulation = reproduction(selectionPopulation);
        offspringPopulation = evaluatePopulation(offspringPopulation);
        population = replacement(population, offspringPopulation);
        updateProgress();
    }

    this.step++;
}

```

Figura 4: Método *runASingleStep* para los algoritmos evolutivos.

2. Como agregar un nuevo problema

Para agregar un nuevo problema se debe implementar la interfaz *Problem* que se muestra en la Figura 1. Esta interfaz posee los siguientes métodos:

- *getNumberOfVariables*: Indica el número de variables del problema.
- *getNumberOfObjectives*: Indica el número de objetivos del problema.
- *getNumberOfConstrains*: Indica el número de restricciones del problema.
- *evaluate*: Evalúa una solución y sus restricciones.

- *createSolution*: Crea una nueva solución para el problema.
- *getLowerBound*: Indica el valor mínimo que puede tomar una variable en un índice específico.
- *getUpperBound*: Indica el valor máximo que puede tomar una variable en un índice específico.
- *applySolutionToNetwork*: Toma una solución y la aplica sobre una red (Instancia de *Network* que posteriormente puede ser guardada como un archivo inp). Este método es opcional y en caso de que no esté implementado devuelve el valor *null*. Se puede ver un ejemplo de este método en la Figura 5. Este método es llamado por la aplicación cuando se selecciona una solución en la pestaña de resultados y se presiona el botón guardar como inp.
- *getName*: El nombre del problema.
- *closeResources*: Cierra los recursos usado por el problema. Generalmente, el único recurso a cerrar sería el simulador hidráulico EpanetAPI. Implementar este método es opcional. Este método es llamado por la aplicación cuando se termina el experimento.

```

public Network applySolutionToNetwork(Network network, Solution<?> solution) {
    // El objeto solucion posee indices que van desde 1 hasta la cantidad de diámetros posibles.
    IntegerSolution iSolution = (IntegerSolution) solution;
    Collection<Link> links = network.getLinks();
    int i = 0;
    for (Link link : links) {
        /*
         * Obtiene el diámetro correspondiente a un indice específico.
         */
        double diameter = this.gamas.get(iSolution.getVariable(i) - 1).getDiameter();
        /*
         * Reemplaza en la tubería y válvula el diámetro. Si en enlace es una bomba se ignora.
         */
        if (link instanceof Pipe) {
            Pipe pipe = (Pipe) link;
            pipe.setDiameter(diameter);
        } else if (link instanceof Valve) {
            Valve valve = (Valve) link;
            valve.setDiameter(diameter);
        }
        i++;
    }
    return network;
}

```

Figura 5: Ejemplo del método *applySolutionToNetwork*.

3. Como agregar un nuevo operador

Para crear un nuevo operador se debe implementar la interfaz *Operator* o alguna de sus subinterfaces o subclases. Esta interfaz cuenta con un único método. La interfaz se muestra en la Figura 1. El método de la interfaz es:

- execute: Método que realiza una operación sobre un operando y devuelve un objeto resultante de dicha operación. Generalmente, este método realiza una acción sobre una solución o una lista de soluciones y retorna la solución resultante de la operación realizada.

Para configurar los parámetros del operador desde la ventana de configuración, debe haber un constructor público que posea la anotación *@DefaultConstructor*.

3.1. *@DefaultConstructor*

La anotación *@DefaultConstructor* indica el constructor que debe ser usado al momento de crear una instancia del operador. Esta anotación recibe un arreglo de *@NumberInput*, el cual se define en la siguiente sección. El arreglo debe tener la misma cantidad de argumentos que los parámetros del constructor como se muestra en la Figura 6 y Figura 7.

```
@DefaultConstructor(@NumberInput(displayName = "constant", defaultValue = 1.5))
public UniformSelection(double constant)
```

Figura 6: Constructor de un solo parámetro.

```
@DefaultConstructor({
    @NumberInput(displayName = "MutationProbability", defaultValue = 0.1),
    @NumberInput(displayName = "DistributionIndex", defaultValue = 0.1
})
public IntegerPolynomialMutation(double mutationProbability, double distributionIndex)
```

Figura 7: Constructor con varios parámetros.

Esta anotación solo puede ser usada en un único constructor por clase. Usar esta anotación en más de un constructor lanzara una excepción en tiempo de ejecución.

Adicionalmente, el constructor que use esta anotación solo puede tener parámetros de tipo *int* o *double*.

La interfaz gráfica creada a partir de las anotaciones de la Figura 6 y 7 se puede ver en la Figura 8 y 9, respectivamente.



Figura 8: Interfaz para configurar el operador *UniformSelection*.

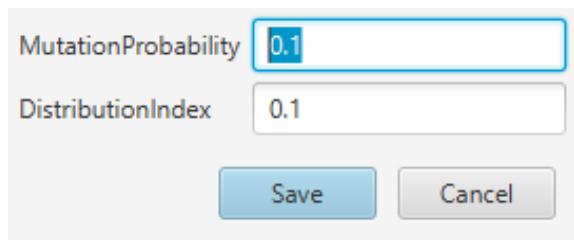


Figura 9: Interfaz para configurar el operador *IntegerPolynomialMutation*.

La anotación puede tener un arreglo vacío, lo cual indica que el constructor no recibirá parámetros.

4. Como agregar un nuevo experimento

Un experimento es una clase que contiene una lista de algoritmos para resolver un problema específico. Los algoritmos que conforman el experimento deben ser de un mismo tipo, por ejemplo, “n” repeticiones del Algoritmo Genético. Actuando “n” como el número de ejecuciones independientes del algoritmo. En la Figura 10, se muestra las interfaces del módulo de metaheurísticas extendido.

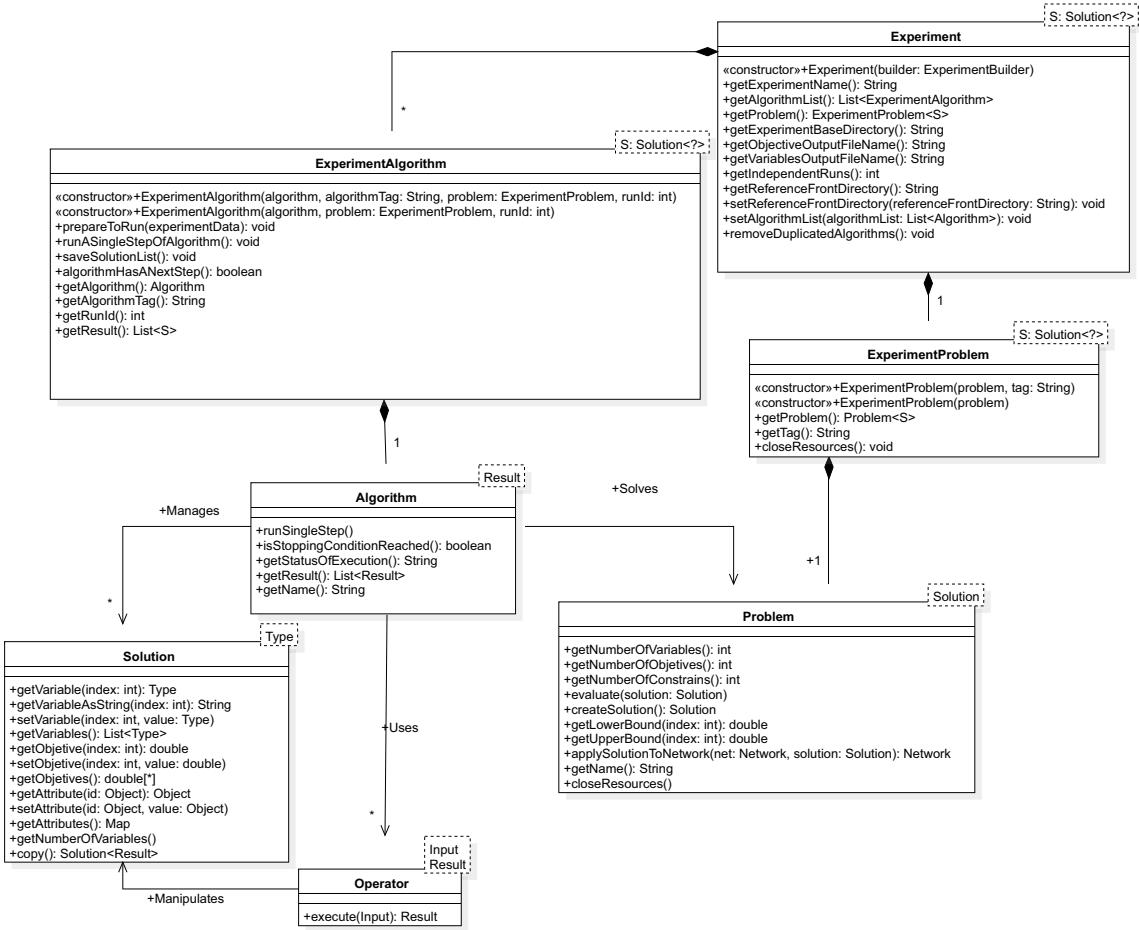


Figura 10: Diagrama módulo metaheurística extendido.

Para crear el experimento la aplicación cuenta con una clase llamada *ExperimentBuilder*. Los parámetros *ExperimentBaseDirectory* y *ReferenceFrontDirectory* solo tienen uso en experimentos multiobjetivos. *ExperimentBaseDirectory* indica donde guardar la frontera de Pareto de cada repetición del algoritmo multiobjetivo. En cambio, *ReferenceFrontDirectory* indica donde guardar la frontera de Pareto final, que se obtiene al unir las fronteras de cada repetición y extraer solamente las soluciones no dominadas de dicho conjunto.

4.1. Interfaz Registrable

Con el fin de agregar un nuevo experimento en la aplicación, se debe crear una clase que herede de *SingleObjectiveRegistrable* o *MultiObjectiveRegistrable*, dependiendo de si el problema es monoobjetivo o multiobjetivo, respectivamente. Las clases que heredan de *Registrable* funcionan como *templates* para crear nuevos experimentos. En la Figura 11, se presenta la jerarquía de clases de la interfaz *Registrable*.

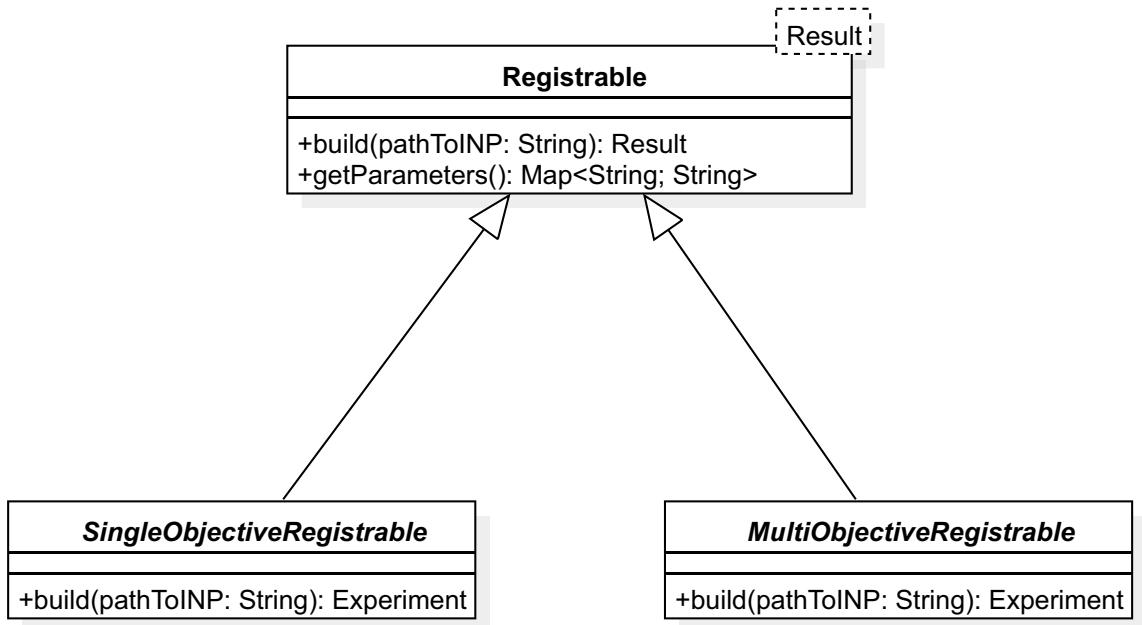


Figura 11: Jerarquía de clases de la interfaz Registrable.

Ambas interfaces devuelven como resultado una instancia del experimento la cual es usada para realizar la simulación. Los métodos utilizados por la interfaz *Registrable* son:

- *build*: Construye un experimento.
- *getParameters*: Este método devuelve un par clave valor, el cual es usado para agregar columnas extras a la ventana de resultados. La clave es usada como el nombre de la columna, mientras que el valor es usado como el valor de la columna. Este método es opcional.

En el constructor público de la clase que herede de una de las subinterfaces de *Registrable*, se debe utilizar la anotación `@NewProblem`. Adicionalmente, si se quiere mostrar la interfaz de configuración, dicho constructor debe incluir la anotación `@Parameters`.

Las clases que implementen cualquiera de las dos interfaces mencionadas anteriormente deben ser guardados en una estructura de datos, la cual es recorrida cuando se inicie la ejecución del programa y analizada usando la *Java Reflection API*. Este análisis consistirá en escanear y validar el cumplimiento de la convención establecida para las clases que implementan esta interfaz. Esta convención consiste en lo siguiente:

- La clase debe contener un único constructor que use la anotación `@NewProblem`.
- Si el constructor requiere parámetros éstos deben estar descritos usando la anotación `@Parameters`.
- El constructor debe declarar los parámetros en el siguiente orden, de acuerdo con su tipo.
 1. *Object*: Usado para injectar los operadores. Estos parámetros pueden posteriormente ser casteados a su tipo correcto. La anotación correspondiente es `@OperatorInput`.
 2. *File*: Usados cuando el problema requiere información adicional que se encuentra en un archivo diferente. La anotación correspondiente es `@FileInput`.
 3. *int, Integer, double o Double*: Usado generalmente para configurar valores en el algoritmo o si el problema requiere otros valores que no fueron solicitados al crear los operadores. Las anotaciones correspondientes son `@NumberInput` y `@NumberToggleInput`.
 4. El constructor debe solicitar la misma cantidad de parámetros que las descritas en la anotación `@Parameters`.

Si estas convenciones no se cumplen, entonces un error en tiempo de compilación será emitido.

El orden en el que son inyectados los parámetros consiste en el siguiente:

1. Parámetros descritos por `@OperatorInput`.
2. Parámetros descritos por `@FileInput`.
3. Parámetros descritos por `@NumberInput`.
4. Parámetros descritos por `@NumberToggleInput`.

Una vez que se haya configurado el problema a través de la interfaz, se creará la instancia de la clase que hereda de `Registrable` y se llamará a su método `build`, para crear el experimento y comenzar su ejecución.

La estructura de datos para registrar las clases que heredan de `SingleObjectiveRegistrable` y `MonoObjectiveRegistrable` se encontrará en la clase `RegistrableConfiguration`.

En las Figura 12, 13, 14, se muestran un ejemplo del constructor, uno del método `build` y uno del método `getParameters`, utilizado en el problema de optimización de los costos de las tuberías.

```

@NewProblem(displayName = "Pipe optimizing", algorithmName = "Genetic Algorithm",
            description = "The objective of this problem is to optimize the cost of "
            "construction of the network by varying the diameter of the pipe in order "
            "to ensure a minimum level of pressure.")
@Parameters(operators = {
    @OperatorInput(displayName = "Selection Operator", value = {
        @OperatorOption(displayName = "Uniform Selection", value = UniformSelection.class)
    }),
    @OperatorInput(displayName = "Crossover Operator", value = {
        @OperatorOption(displayName = "Integer Single Point Crossover", value = IntegerSinglePointCrossover.class),
        @OperatorOption(displayName = "Integer SBX Crossover", value = IntegerSBXCrossover.class)
    }),
    // @OperatorInput(displayName = "Mutation Operator", value = {
    //     @OperatorOption(displayName = "Integer Simple Random Mutation", value = IntegerSimpleRandomMutation.class),
    //     @OperatorOption(displayName = "Integer Polynomial Mutation", value = IntegerPolynomialMutation.class),
    //     @OperatorOption(displayName = "Integer Range Random Mutation", value = IntegerRangeRandomMutation.class)
    // }), //
    files = {
        @FileInput(displayName = "Gama")
    },
    // numbers = {
    //     @NumberInput(displayName = "Independent run", defaultValue = 5),
    //     @NumberInput(displayName = "Min pressure", defaultValue = 30),
    //     @NumberInput(displayName = "Population Size", defaultValue = 1000)
    // },
    numbersToggle = {
        @NumberToggleInput(groupID = "Finish Condition", displayName = "Number of iteration without improvement", defaultValue = 100),
        @NumberToggleInput(groupID = "Finish Condition", displayName = "Max number of evaluation", defaultValue = 1000)
    }
})
public PipeOptimizingRegister(Object selectionOperator, Object crossoverOperator, Object mutationOperator, File gama, int independentRun,
                               int minPressure, int populationSize, int numberWithoutImprovement, int maxEvaluations) throws Exception

```

Figura 12: Constructor de la clase `PipeOptimizingRegister`.

```

@Override
public Experiment<IntegerSolution> build(String inpPath) throws Exception {
    if (inpPath == null || inpPath.isEmpty()) {
        throw new ApplicationException("There isn't a network opened");
    }
    // Crea el simulador hidráulico.
    EpanetAPI epanet = new EpanetAPI();
    /*
     * Inicializa el simulador. Es importante no llamar a epanet.ENclose() hasta que termina la ejecución
     * del experimento. Es por ello, que la llamada a epanet.ENClose() debe ir en el método closeResource del problema,
     * puesto que este método se llama cuando termina la ejecución del algoritmo.
     */
    epanet.ENopen(inpPath, "ejecucion.rpt", "");

    // El archivo gama, es un archivo que posee los diámetros disponibles de la red y el costo de éstos.
    if (this.gama == null) {
        throw new ApplicationException("There isn't gama file");
    }
    this.problem = new PipeOptimizing(epanet, this.gama.getAbsolutePath(), this.minPressure);

    // Indica el problema a usar por el experimento
    ExperimentProblem<IntegerSolution> experimentProblem = new ExperimentProblem<>(this.problem);

    /*
     * Crea varios repeticiones del mismo algoritmo. En este caso, son del algoritmos genético.
     * El número de algoritmos genéticos a crear corresponde al valor de la variable independentRun.
     */
    List<ExperimentAlgorithm<IntegerSolution>> experimentAlgorithms = ExperimentUtils.configureAlgorithmList(experimentProblem,
        this.independentRun,
        () -> {
            GeneticAlgorithmBuilder builder = new GeneticAlgorithmBuilder(this.problem, this.crossover, this.mutation)
                .setCrossoverOperator(crossover);
            if (this.numberWithoutImprovement != Integer.MIN_VALUE) {
                builder.setMaxNumberOfEvaluationWithoutImprovement(this.numberWithoutImprovement);
            } else {
                builder.setMaxEvaluations(this.maxEvaluations);
            }
            return builder.build();
        });
    // Configura el experimento
    Experiment<IntegerSolution> experiment = new ExperimentBuilder<IntegerSolution>("PipeOptimizing")
        .setIndependentRuns(this.independentRun)
        .setAlgorithmList(experimentAlgorithms)
        .setProblem(experimentProblem)
        .build();

    return experiment;
}

```

Figura 13: Implementación del método build de la clase *PipeOptimizingRegister*.

```

@Override
public Experiment<IntegerSolution> build(String inpPath) throws Exception {
    if (inpPath == null || inpPath.isEmpty()) {
        throw new ApplicationException("There isn't a network opened");
    }
    // Crea el simulador hidráulico.
    EpanetAPI epanet = new EpanetAPI();
    /*
     * Inicializa el simulador. Es importante no llamar a epanet.ENClose() hasta que termina la ejecución
     * del experimento. Es por ello, que la llamada a epanet.ENClose() debe ir en el método closeResource del problema,
     * puesto que este método se llama cuando termina la ejecución del algoritmo.
     */
    epanet.ENOpen(inpPath, "ejecucion.rpt", "");

    // El archivo gama, es un archivo que posee los diámetros disponibles de la red y el costo de éstos.
    if (this.gama == null) {
        throw new ApplicationException("There isn't gama file");
    }
    this.problem = new PipeOptimizing(epanet, this.gama.getAbsolutePath(), this.minPressure);

    // Indica el problema a usar por el experimento
    ExperimentProblem<IntegerSolution> experimentProblem = new ExperimentProblem<>(this.problem);

    /*
     * Crea varios repeticiones del mismo algoritmo. En este caso, son del algoritmos genético.
     * El número de algoritmos genéticos a crear corresponde al valor de la variable independentRun.
     */
    List<ExperimentAlgorithm<IntegerSolution>> experimentAlgorithms = ExperimentUtils.configureAlgorithmList(experimentProblem,
        this.independentRun,
        () -> {
            GeneticAlgorithmBuilder builder = new GeneticAlgorithmBuilder(this.problem, this.crossover, this.mutation)
                .setCrossoverOperator(crossover);
            if (this.numberWithoutImprovement != Integer.MIN_VALUE) {
                builder.setMaxNumberOfEvaluationWithoutImprovement(this.numberWithoutImprovement);
            } else {
                builder.setMaxEvaluations(this.maxEvaluations);
            }
            return builder.build();
        });
    // Configura el experimento
    Experiment<IntegerSolution> experiment = new ExperimentBuilder<IntegerSolution>("PipeOptimizing")
        .setIndependentRuns(this.independentRun)
        .setAlgorithmList(experimentAlgorithms)
        .setProblem(experimentProblem)
        .build();

    return experiment;
}

```

Figura 14: Implementación del método opcional `getParameters` de la clase `PipeOptimizingRegister`.

A continuación se define detalladamente cada una de las anotaciones permitidas en la clase Registrable.

4.2. `@NewProblem`

Esta anotación permite indicar el nombre del problema que será mostrado en la interfaz gráfica. Puedes ver el uso de esta anotación en la Figura 15.

```

@NewProblem(displayName = "Pipe optimizing", algorithmName = "Genetic Algorithm",
    description = "The objective of this problem is to optimize the cost of "
        "construction of the network by varying the diameter of the pipe in order "
        "to ensure a minimum level of pressure.")
@Parameters(operators = {
    @OperatorInput(displayName = "Selection Operator", value = {
        @OperatorOption(displayName = "Uniform Selection", value = UniformSelection.class)
    }),
    @OperatorInput(displayName = "Crossover Operator", value = {
        @OperatorOption(displayName = "Integer Single Point Crossover", value = IntegerSinglePointCrossover.class),
        @OperatorOption(displayName = "Integer SBX Crossover", value = IntegerSBXCrossover.class)
    })
}, // ...
@OperatorInput(displayName = "Mutation Operator", value = {
    @OperatorOption(displayName = "Integer Random Mutation", value = IntegerSimpleRandomMutation.class),
    @OperatorOption(displayName = "Integer Polynomial Mutation", value = IntegerPolynomialMutation.class),
    @OperatorOption(displayName = "Integer Range Random Mutation", value = IntegerRangeRandomMutation.class)
}), // ...
files = {
    @FileInput(displayName = "Gama")
}, // ...
numbers = {
    @NumberInput(displayName = "Independent run", defaultValue = 5),
    @NumberInput(displayName = "Min Pressure", defaultValue = 30),
    @NumberInput(displayName = "Population Size", defaultValue = 1000)
}, // ...
numbersToggle = {
    @NumberToggleInput(groupID = "Finish Condition", displayName = "Number of iteration without improvement", defaultValue = 100),
    @NumberToggleInput(groupID = "Finish Condition", displayName = "Max number of evaluation", defaultValue = 1000)
}
)
public PipeOptimizingRegister(Object selectionOperator, Object crossoverOperator, Object mutationOperator, File gama, int independentRun,
    int minPressure, int populationSize, int numberWithoutImprovement, int maxEvaluations) throws Exception

```

Figura 15: Constructor de clase que hereda de registrable y sus metadatos para cada parámetro.

Los elementos en esta anotación consisten en:

- *displayName*: El nombre del problema. Este nombre también actúa como el nombre de la categoría.
- *algorithm*: Un *String* con el nombre del algoritmo usado para resolver el problema.
- *description*: Un *String* con la descripción del algoritmo.

El nombre dado en el elemento *displayName*, es el nombre visible del problema en el menú de la aplicación y permite agrupar a los problemas que tengan el mismo nombre, pero distintos algoritmos, como se ve en la Figura 16.

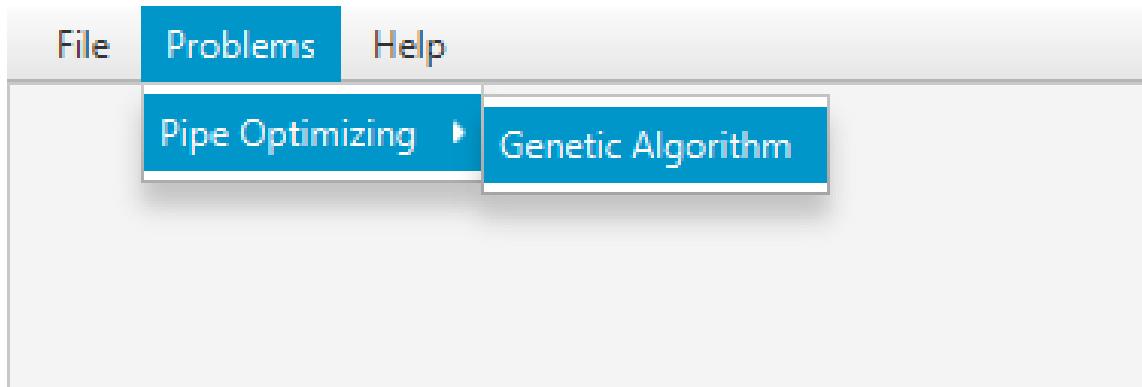


Figura 16: Menú de problemas.

Esta anotación solo puede estar en un constructor, en caso de que esta anotación no esté presente, un error en tiempo de ejecución será lanzado.

4.3. *@Parameters*

Esta anotación permite agregar información acerca de los parámetros recibidos por el constructor. Cuando el constructor tiene esta anotación, por convención, está obligado a declarar los parámetros en un orden determinado en base a tu tipo. Este orden es el siguiente:

1. *Object*

2. *File*
3. *int o double*

Si el constructor no declara los parámetros en ese orden un error en tiempo de ejecución será lanzado.

Dentro de esta anotación existen varios elementos. Estos elementos son:

- *operators*: Arreglo que recibe anotaciones del tipo *OperatorInput*.
- *files*: Arreglo que recibe anotaciones del tipo *FileInput*.
- *numbers*: Arreglo que recibe anotaciones del tipo *NumberInput*.
- *numbersToggle*: Arreglo que recibe anotaciones del tipo *NumberToggleInput*.

El valor por defecto para todos los elementos mencionados anteriormente es un arreglo vacío ({}). No usar la anotación *@Parameters* tiene el mismo efecto que usar la anotación, pero con sus valores por defecto.

Puede ver el uso de esta anotación en la Figura 15.

4.4. *@OperatorInput*

Esta anotación agrega información a uno de los parámetros del constructor acerca de los posibles operadores que pueden ser recibidos por ese parámetro. Dentro de esta anotación existen varios elementos. Estos elementos son:

1. *displayName*: Nombre de categoría para los operadores.
2. *value*: Arreglo que recibe anotaciones del tipo *OperatorOption*.

En la ventana de configuración de problema, estos parámetros son vistos con un *ComboBox* como muestra la Figura 17.

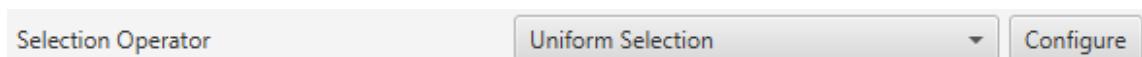


Figura 17: Componente *ComboBox* para configurar los operadores.

Las alternativas disponibles dentro del *ComboBox* están dadas por aquellas indicadas en el elemento *value* de este operador. Como se muestra en la Figura 15, la única alternativa para el *Selection Operator* es el operador *UniformSelection* apreciado en la Figura 18.

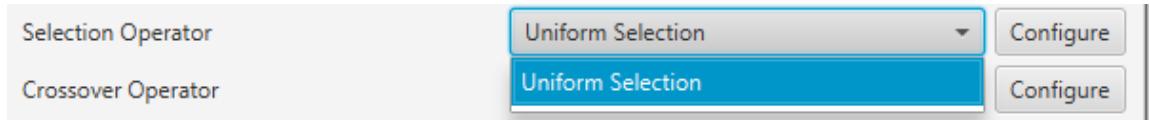


Figura 18: *ComboBox* expandido para configurar el operador.

Por defecto, el *ComboBox* selecciona el primer elemento de la lista. El botón *Configure* permite configurar los parámetros que recibe el constructor del operador, aquel que posee la anotación *@DefaultConstructor*. Para el caso del operador *UniformSelection*, su interfaz de configuración se muestra en la Figura 8.

4.5. *@OperatorOption*

Esta anotación permite indicar las alternativas de operadores que puede recibir un parámetro para una categoría de operador indicada por la anotación *@OperatorInput*. Dentro de esta anotación existen varios elementos. Estos elementos son:

- *displayName*: Nombre del operador. Este es el nombre visualizado en el *ComboBox* como se muestra en la Figura 18.
- *value*: Instancia del tipo *Class* que referencia el tipo de operador.

4.6. *@FileInput*

Esta anotación indica que hay un parámetro que espera recibir un objeto de tipo *File*. Cuando esta anotación está presente junto con su parámetro, en la interfaz, aparecerá un apartado que abre un *FileChooser* o un *DirectoryChooser* para buscar un archivo o directorio, respectivamente. Dentro de esta anotación existen varios elementos. Éstos son:

- *displayName*: Nombre del parámetro. Este nombre también corresponde al nombre visualizado en la ventana de configuración como se muestra en la Figura 19.

- *type*: Indica el modo en que se abrirá el *FileChooser*. Este elemento recibe un enumerado del tipo *Type*; los cuales son *Type.OPEN*, *Type.SAVE*, que abren un *FileChooser* para leer o guardar un archivo; y *Type.Directory*, el cual abre un *DirectoryChooser* para seleccionar un directorio. La opción por defecto es *FileType.OPEN*.

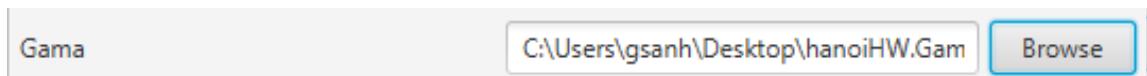


Figura 19: Apartado para configurar el parámetro de tipo *File*.

Si el *TextField* donde se muestra la ruta está vacío, es decir, no se ha seleccionado un archivo o carpeta, entonces será inyectado ***null*** en el parámetro correspondiente del constructor.

4.7. *@NumberInput*

Esta anotación indica que hay un parámetro del tipo *int* o *double* o sus tipos envoltorios *Integer* o *Double*, respectivamente. Esta anotación agrega en la interfaz un *TextField* que solo permite como entrada un número. Si el tipo del parámetro es *int* o *Integer*, entonces el *TextField* solo permite ingresar números enteros. Por otro lado, si el parámetro es *double* o *Double*, entonces en la interfaz se acepta el ingreso de números enteros o decimales. El apartado para esta anotación se muestra en la Figura 20.



Figura 20: *TextField* presente cuando está la anotación *@NumberInput*.

Dentro de esta anotación existen los siguientes elementos:

- *displayName*: Nombre del parámetro.
- *defaultValue*: Valor por defecto de la propiedad. Si el tipo de parámetro en el constructor de la clase que hereda de *Registrable* es un entero, pero se ingresa

como valor por defecto un número con decimales, los decimales serán truncados. Si este elemento no se define su valor por defecto es 0.

4.8. *@NumberToggleInput*

Esta anotación indica que hay un conjunto de parámetros que son mutuamente excluyentes entre ellos, es decir, que solo un parámetro puede recibir el valor. En la interfaz, el nombre del grupo aparece sobre los componentes. Dentro de un mismo grupo solo se puede configurar un parámetro. El parámetro por configurar debe ser indicado activando el *ToggleButton* correspondiente, lo cual conlleva a la activación del *TextField*. Dentro de esta anotación existen varios elementos. Estos elementos son:

- *groupID*: *String* con un id para el grupo. Las anotaciones *NumberToggerInput* que tengan el mismo id, en la interfaz, se encuentran en una sección cuyo título es el nombre del grupo. Esto se aprecia en la Figura 21.
- *displayName*: Nombre del parámetro.
- *defaultValue*: Valor por defecto de la propiedad. Si el tipo de parámetro en el constructor de la clase que hereda de *Registrable* es un entero, pero se ingresa como valor por defecto un número con decimales, los decimales serán truncados. Si este elemento no se define su valor por defecto es 0.



Figura 21: Apartado para *NumberToggleInput* con el mismo GroupID.

El parámetro configurado en la interfaz de usuario recibe el valor indicado en el *TextField*. Si el *TextField* queda vacío entonces recibe el valor cero. Sin embargo, los demás parámetros, cuyos *TextField* están deshabilitados, van a recibir el

valor ***Double.MIN_VALUE***, si el parámetro es de tipo *double* o *Double*; o ***Integer.MIN_VALUE*** si el parámetro es de tipo *int* o *Integer*. Por ejemplo, en la Figura 21, se observa que el parámetro “*Number of iteration without improvement*” esta activado, pero no contiene un valor, entonces al crear la instancia el constructor va a recibir el valor cero. Pero el parámetro “*Max number of evaluation*”, al no haber sido escogido, recibe el valor *Integer.MIN_VALUE*, puesto que este parámetro era de tipo *int* o *Integer*.

En la Figura 15 se puede ver el uso de la anotación *NumberToggleInput*. El componente que representa esta anotación en la GUI se puede apreciar en la Figura 21. El groupID es usado para nombrar la sección.

En el elemento *numbersToggle* de la anotación *@Parameters*, las anotaciones que pertenezcan al mismo grupo deben estar continuas. En caso de que esto no se cumpla se lanza una excepción al momento de ejecutar la aplicación.

Las anotaciones presentadas en las dos secciones anteriores deben ser usadas en constructores públicos.

Glosario

GA Algoritmo Genético

NSGAII Non-dominated Sorting Genetic Algorithm

D. Documento de casos de prueba



UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Documento de especificación de casos de prueba

**HERRAMIENTA PARA LA OPTIMIZACIÓN DE REDES DE
DISTRIBUCIÓN DE AGUA POTABLE**

TABLA DE CONTENIDOS

página

Tabla de Contenidos

1. Casos de pruebas automatizadas	1
1.1. <i>Pump</i>	1
1.2. <i>GeneticAlgorithm</i>	8
1.3. <i>IntegerRangeRandomMutation</i>	10
1.4. <i>ReflectionUtils</i>	13
1.5. <i>ResultSimutation</i>	22
1.6. <i>JsonSimpleReader</i>	24
2. Casos de pruebas manuales	32

1. Casos de pruebas automatizadas

En este capítulo se presenta la especificación formal de los casos de prueba automatizados. Los casos de prueba automatizados son aquellos que se realizan utilizando herramientas de automatización como el framework JUnit y que no necesitan interacción con un usuario durante su ejecución.

1.1. *Pump*

En esta sección se presentan las pruebas realizadas para la clase *Pump*.

Test ID:	AT001
Título:	Envío de parámetro inválido al método <code>setProperty</code> de la clase <code>Pump</code> cuando se usa la clave <code>PumpProperty.HEAD</code> .
Característica:	Validar parámetro recibido por el método <code>setProperty</code> .
Objetivo:	Comprobar que al pasar un parámetro como valor de un tipo distinto a <code>String</code> , cuando se usa la clave <code>PumpProperty.HEAD</code> , se lanza una excepción.
Configuración:	Instancia de la clase <code>Pump</code> inicializada.
Datos de prueba:	Clave: <code>PumpProperty.HEAD</code> Valor: Un objeto diferente a un <code>String</code>
Acciones de prueba:	1. Inicializar instancia. 2. Pasar una instancia de un objeto distinto de un <code>String</code> al método cuando se usa la llave <code>PumpProperty.HEAD</code> .
Resultados esperados:	Excepción indicando que el tipo de instancia pasada no es válida.

Test ID:	AT002
Título:	Envío de parámetro inválido al método <code>setProperty</code> de la clase <code>Pump</code> cuando se usa la clave <code>PumpProperty.PATTERN</code> .
Característica:	Validar parámetro recibido por el método <code>setProperty</code> .
Objetivo:	Comprobar que si se pasa un parámetro como valor de un tipo distinto a <code>String</code> , cuando se usa la clave <code>PumpProperty.PATTERN</code> , se lanza una excepción.
Configuración:	Instancia de la clase <code>Pump</code> inicializada.
Datos de prueba:	Clave: <code>PumpProperty.PATTERN</code> Valor: Un objeto diferente a un <code>String</code>
Acciones de prueba:	1. Inicializar instancia. 2. Pasar una instancia de un objeto distinto de un <code>String</code> al método cuando se usa la llave <code>PumpProperty.HEAD</code> .
Resultados esperados:	Excepción indicando que el tipo de instancia pasada no es válida.

Test ID:	AT003
Título:	Envío de parámetro inválido al método <code>setProperty</code> de la clase <code>Pump</code> cuando se usa la clave <code>PumpProperty.SPEED</code> .
Característica:	Validar parámetro recibido por el método <code>setProperty</code> .
Objetivo:	Comprobar que si se pasa un parámetro como valor de un tipo distinto a <code>Double</code> , cuando se usa la clave <code>PumpProperty.SPEED</code> , el método lanza una excepción.
Configuración:	Instancia de la clase <code>Pump</code> inicializada.
Datos de prueba:	Clave: <code>PumpProperty.SPEED</code> Valor: Un <code>Integer</code> o alguna instancia de otro objeto distinto de <code>Double</code> .
Acciones de prueba:	1. Inicializar instancia. 2. Pasar una instancia de un objeto distinto de un <code>Double</code> al método cuando se usa la llave <code>PumpProperty.SPEED</code> .
Resultados esperados:	Excepción indicando que el tipo de instancia pasada no es válida.

Test ID:	AT004
Título:	Envío de parámetro inválido al método <code>setProperty</code> de la clase <code>Pump</code> cuando se usa la clave <code>PumpProperty.POWER</code> .
Característica:	Validar parámetro recibido por el método <code>setProperty</code> .
Objetivo:	Comprobar que si se pasa un parámetro como valor de un tipo distinto a <code>Double</code> , cuando se usa la clave <code>PumpProperty.POWER</code> , el método lanza una excepción.
Configuración:	Instancia de la clase <code>Pump</code> inicializada.
Datos de prueba:	Clave: <code>PumpProperty.POWER</code> Valor: Un <code>Integer</code> o alguna instancia de otro objeto distinto de <code>Double</code> .
Acciones de prueba:	1. Inicializar instancia. 2. Pasar una instancia de un objeto distinto de un <code>Double</code> al método cuando se usa la llave <code>PumpProperty.POWER</code> .
Resultados esperados:	Excepción indicando que el tipo de instancia pasada no es válida.

Test ID:	AT005
Título:	Envío de parámetro válido al método <i>setProperty</i> de la clase <i>Pump</i> cuando se usa la clave <i>PumpProperty.HEAD</i> .
Característica:	Validar parámetro recibido por el método <i>setProperty</i> .
Objetivo:	Comprobar que si se pasa un parámetro de tipo <i>String</i> como valor, cuando se usa la clave <i>PumpProperty.HEAD</i> , el método finaliza sin error.
Configuración:	Instancia de la clase <i>Pump</i> inicializada.
Datos de prueba:	Clave: <i>PumpProperty.HEAD</i> Valor: Un <i>String</i> no vacío.
Acciones de prueba:	1. Inicializar instancia. 2. Pasar una instancia de un <i>String</i> al método cuando se usa la llave <i>PumpProperty.HEAD</i> .
Resultados esperados:	Método ejecutado sin errores.

Test ID:	AT006
Título:	Envío de parámetro válido al método <i>setProperty</i> de la clase <i>Pump</i> cuando se usa la clave <i>PumpProperty.PATTERN</i> .
Característica:	Validar parámetro recibido por el método <i>setProperty</i> .
Objetivo:	Comprobar que si se pasa un parámetro de tipo <i>String</i> como valor, cuando se usa la clave <i>PumpProperty.PATTERN</i> , el método finaliza sin error.
Configuración:	Instancia de la clase <i>Pump</i> inicializada.
Datos de prueba:	Clave: <i>PumpProperty.PATTERN</i> Valor: Un <i>String</i> no vacío.
Acciones de prueba:	1. Inicializar instancia. 2. Pasar una instancia de un <i>String</i> al método cuando se usa la llave <i>PumpProperty.HEAD</i> .
Resultados esperados:	Método ejecutado sin errores.

Test ID:	AT007
Título:	Envío de parámetro válido al método <i>setProperty</i> de la clase <i>Pump</i> cuando se usa la clave <i>PumpProperty.SPEED</i> .
Característica:	Validar parámetro recibido por el método <i>setProperty</i> .
Objetivo:	Comprobar que si se pasa un parámetro de tipo <i>Double</i> como valor, cuando se usa la clave <i>PumpProperty.SPEED</i> , el método finaliza sin error.
Configuración:	Instancia de la clase <i>Pump</i> inicializada.
Datos de prueba:	Clave: <i>PumpProperty.SPEED</i> Valor: Un valor <i>Double</i> .
Acciones de prueba:	1. Inicializar instancia. 2. Pasar una instancia de un <i>Double</i> al método cuando se usa la llave <i>PumpProperty.SPEED</i> .
Resultados esperados:	Método ejecutado sin errores.

Test ID:	AT008
Título:	Envío de parámetro válido al método <i>setProperty</i> de la clase <i>Pump</i> cuando se usa la clave <i>PumpProperty.POWER</i> .
Característica:	Validar parámetro recibido por el método <i>setProperty</i> .
Objetivo:	Comprobar que si se pasa un parámetro de tipo <i>Double</i> como valor, cuando se usa la clave <i>PumpProperty.POWER</i> , el método finaliza sin error.
Configuración:	Instancia de la clase <i>Pump</i> inicializada.
Datos de prueba:	Clave: <i>PumpProperty.POWER</i> Valor: Un valor <i>Double</i> .
Acciones de prueba:	1. Inicializar instancia. 2. Pasar una instancia de un <i>Double</i> al método cuando se usa la llave <i>PumpProperty.SPEED</i> .
Resultados esperados:	Método ejecutado sin errores.

1.2. *GeneticAlgorithm*

En esta sección se especifican las pruebas realizadas para la clase *GeneticAlgorithm*.

Test ID:	AT009
Título:	Número máximo de evaluaciones no válido.
Característica:	Validar parámetro <i>maxEvaluations</i> .
Objetivo:	Validar que el parámetro <i>maxEvaluations</i> no sea negativo. Si el parámetro es negativo debe lanzarse una excepción.
Configuración:	Instancia de la clase <i>GeneticAlgorithm</i> inicializada.
Datos de prueba:	Cualquier entero menor que 0.
Acciones de prueba:	Llamar al método <i>setMaxEvaluations</i> con un argumento negativo.
Resultados esperados:	Una excepción.

Test ID:	AT010
Título:	Número máximo de evaluaciones sin mejora no válido.
Característica:	Validar parámetro <i>maxNumberOfEvaluationsWithoutImprovement</i> .
Objetivo:	Validar que el parámetro <i>maxNumberOfEvaluationsWithoutImprovement</i> no sea negativo. Si el parámetro es negativo debe lanzarse una excepción.
Configuración:	Instancia de la clase <i>GeneticAlgorithm</i> inicializada.
Datos de prueba:	Cualquier entero menor que 0.
Acciones de prueba:	Llamar al método <i>setMaxNumberOfEvaluationsWithoutImprovement</i> con un argumento negativo.
Resultados esperados:	Una excepción.

Test ID:	AT011
Título:	Deshabilitar número máximo de evaluaciones sin mejoras cuando se modifica el número máximo de evaluaciones.
Característica:	Validar parámetro <i>maxEvaluations</i> .
Objetivo:	Validar que al modificar el parámetro <i>maxEvaluations</i> el parámetro <i>maxNumberOfEvaluationsWithoutImprovement</i> cambie a 0 (0 indica que esta deshabilitado).
Configuración:	Instancia de la clase <i>GeneticAlgorithm</i> inicializada.
Datos de prueba:	Cualquier entero positivo.
Acciones de prueba:	Llamar al método <i>setMaxEvaluations</i> con un entero positivo mayor a 0.
Resultados esperados:	Parámetro <i>maxNumberOfEvaluationsWithoutImprovement</i> igual a 0.

Test ID:	AT012
Título:	Deshabilitar número máximo de evaluaciones cuando se modifica el número máximo de evaluaciones sin mejoras.
Característica:	Validar parámetro <i>maxNumberOfEvaluationsWithoutImprovement</i> .
Objetivo:	Validar que al modificar el parámetro <i>maxNumberOfEvaluationsWithoutImprovement</i> el parámetro <i>maxNumberOfEvaluationsWithoutImprovement</i> cambie a 0 (0 indica que esta deshabilitado).
Configuración:	Instancia de la clase <i>GeneticAlgorithm</i> inicializada.
Datos de prueba:	Cualquier entero positivo.
Acciones de prueba:	Llamar al método <i>setMaxNumberOfEvaluationsWithoutImprovement</i> con un entero positivo mayor a 0.
Resultados esperados:	Parámetro <i>maxEvaluations</i> igual a 0.

1.3. *IntegerRangeRandomMutation*

En esta sección se especifican las pruebas realizadas sobre la clase *IntegerRangeRandomMutation*.

Test ID:	AT013
Título:	Probabilidad de mutación no válida.
Característica:	Validar parámetro <i>mutationProbability</i> .
Objetivo:	Validar que el parámetro <i>mutationProbability</i> no sea negativo. Si el parámetro es negativo debe lanzarse una excepción.
Configuración:	Instancia de la clase <i>IntegerRangeRandomMutation</i> inicializada.
Datos de prueba:	Cualquier entero menor que 0.
Acciones de prueba:	Crear instancia de la clase <i>IntegerRangeRandomMutation</i> pasando como argumento para el parámetro <i>mutationProbability</i> un valor negativo.
Resultados esperados:	Una excepción.

Test ID:	AT014
Título:	Rango no válido.
Característica:	Validar parámetro <i>range</i> .
Objetivo:	Validar que el parámetro <i>range</i> no sea negativo. Si el parámetro es negativo debe lanzarse una excepción.
Configuración:	Instancia de la clase <i>IntegerRangeRandomMutation</i> inicializada.
Datos de prueba:	Cualquier entero menor que 0.
Acciones de prueba:	Crear instancia de la clase <i>IntegerRangeRandomMutation</i> pasando como argumento para el parámetro <i>range</i> un valor negativo.
Resultados esperados:	Una excepción.

Test ID:	AT015
Título:	Mutar variables.
Característica:	Mutar variables cuando el número generado por el <i>randomGenerator</i> es menor que la probabilidad de mutación.
Objetivo:	Validar que la mutación suceda cuando un número generado aleatoriamente sea menor que la probabilidad de mutación.
Configuración:	Solucion con variables preestablecidas. Operador <i>IntegerRangeRandomMutation</i> inicializado.
Datos de prueba:	<i>mutationProbability</i> : 0.3 <i>range</i> : 2 <i>solution</i> : [0, 2, 4] <i>randomGenerator</i> : [0.2, 0.2, 0.4] <i>boundedRandomGenerator</i> : [2, 0, 5]
Acciones de prueba:	Pasar la solución al método <i>execute</i> .
Resultados esperados:	Las variables en la solución después de realizar la mutación deben ser [2, 0, 4]

Test ID:	AT016
Título:	No mutar variables.
Característica:	No mutar variables cuando los números generados por el <i>randomGenerator</i> sean mayores que la probabilidad de mutación.
Objetivo:	Validar que la mutación no sucede si el <i>randomGenerator</i> devuelve valores mayores a <i>mutationProbability</i> .
Configuración:	Solución con variables preestablecidas. Operador <i>IntegerRangeRandomMutation</i> inicializado.
Datos de prueba:	<i>mutationProbability</i> : 0.3 <i>range</i> : 2 <i>solution</i> : [0, 2, 4] <i>randomGenerator</i> : [0.5, 0.4, 0.4] <i>boundedRandomGenerator</i> : [2, 0, 5]
Acciones de prueba:	Pasar la solución al método <i>execute</i> .
Resultados esperados:	Las variables en la solución después de realizar la llamada al método <i>execute</i> deben ser los valores originales [0, 2, 4].

Test ID:	AT017
Título:	No mutar variables cuando no hay un rango de mutación.
Característica:	No mutar variables cuando el rango de las variables a generar es 0.
Objetivo:	Validar que la mutación no sucede si <i>range</i> tiene asignado el valor 0.
Configuración:	Solucion con variables preestablecidas. Operador <i>IntegerRangeRandomMutation</i> inicializado.
Datos de prueba:	<i>mutationProbability</i> : 0.3 <i>range</i> : 0 <i>solution</i> : [0, 2, 4] <i>randomGenerator</i> : [0.1, 0.2, 0.3]
Acciones de prueba:	Pasar la solución al método <i>execute</i> .
Resultados esperados:	Las variables en la solución después de realizar la llamada al método <i>execute</i> deben ser los valores originales [0, 2, 4].

1.4. *Reflection Utils*

En esta sección se especifican las pruebas realizadas sobre la clase *Reflection Utils*.

Durante esta sección nos referiremos a cualquier implementación de las interfaz *Registrable* o sus subinterfaces únicamente como *Registrable*. Es por ello, que cuando se mencione implementar *Registrable* se refiere a implementar ya sea *SingleObjectiveRegistrable* o *MultiobjectiveRegistrable*.

Test ID:	AT018
Título:	Nombre del problema en anotación <code>@NewProblem</code> .
Característica:	Obtener nombre del problema de la anotación <code>@NewProblem</code> .
Objetivo:	Validar que el método <code>getNameOfProblem</code> retorna el nombre asignado en la anotación <code>@NewProblem</code> .
Configuración:	Implementación de <code>Registrable</code> con la anotación <code>@NewProblem</code> en su constructor.
Datos de prueba:	Nombre del problema: "Test"
Acciones de prueba:	Pasar el objeto <code>Class<?></code> que hace referencia al tipo <code>Registrable</code> al método <code>getNameOfProblem</code> .
Resultados esperados:	"Test"

Test ID:	AT019
Título:	Nombre del algoritmo en anotación <code>@NewProblem</code> .
Característica:	Obtener nombre del algoritmo de la anotación <code>@NewProblem</code> .
Objetivo:	Validar que el método <code>getNameOfAlgorithm</code> retorna el nombre asignado en la anotación <code>@NewProblem</code> .
Configuración:	Implementación <code>Registrable</code> con la anotación <code>@NewProblem</code> en su constructor.
Datos de prueba:	Nombre del algoritmo: "NSGAII"
Acciones de prueba:	Pasar el objeto <code>Class<?></code> que hace referencia al tipo <code>Registrable</code> al método <code>getNameOfAlgorithm</code> .
Resultados esperados:	" NSGAII"

Test ID:	AT020
Título:	<i>Registrable</i> sin anotaciones.
Característica:	Validar las anotaciones y el tipo de parámetros utilizados en el constructor de la clase <i>Registrable</i> .
Objetivo:	Validar que si en el constructor público no tiene ni la anotación <code>@NewProblem</code> ni la anotación <code>@Parameters</code> se lanza una excepción.
Configuración:	Implementación de <i>Registrable</i> sin anotaciones.
Datos de prueba:	Objeto <code>Class<?></code> que referencia al tipo <i>Registrable</i> .
Acciones de prueba:	Pasar el objeto <code>Class</code> que hace referencia al tipo <i>Registrable</i> al método <code>validateRegistrableProblem</code> .
Resultados esperados:	Una excepción.

Test ID:	AT021
Título:	<i>Registrable</i> con la anotación <code>@NewProblem</code> .
Característica:	Validar las anotaciones y el tipo de parámetros utilizados en el constructor de la clase <i>Registrable</i> .
Objetivo:	Validar si en el constructor público está la anotación <code>@NewProblem</code>
Configuración:	Implementación de <i>Registrable</i> con la anotación <code>@NewProblem</code> .
Datos de prueba:	Objeto <code>Class<?></code> que referencia al tipo <i>Registrable</i> .
Acciones de prueba:	Pasar el objeto <code>Class</code> que hace referencia al tipo <i>Registrable</i> al método <code>validateRegistrableProblem</code> .
Resultados esperados:	Método ejecutado sin errores.

Test ID:	AT022
Título:	<i>Registrable</i> cuyo constructor recibe los parámetros en el orden correcto dependiendo de su tipo.
Característica:	Validar las anotaciones y el tipo de parámetros utilizados en el constructor de la clase <i>Registrable</i> .
Objetivo:	Validar que el constructor público con las anotaciones recibe los parámetros en el siguiente orden: <i>Object, File, (int Integer double Double)</i> .
Configuración:	Implementación de <i>Registrable</i> con las anotaciones <i>@New-Problem</i> y <i>@Parameters</i> , y con los parámetros recibidos en el constructor en el orden esperado.
Datos de prueba:	Objeto <i>Class<?></i> que referencia al tipo <i>Registrable</i> .
Acciones de prueba:	Pasar el objeto <i>Class</i> que hace referencia al tipo <i>Registrable</i> al método <i>validateRegistrableProblem</i> .
Resultados esperados:	Método ejecutado sin errores.

Test ID:	AT023
Título:	<i>Registrable</i> cuyo constructor recibe los parámetros en un orden incorrecto.
Característica:	Validar las anotaciones y el tipo de parámetros utilizados en el constructor de la clase <i>Registrable</i> .
Objetivo:	Validar que el constructor público con las anotaciones recibe los parámetros en un orden distinto a: <i>Object</i> , <i>File</i> , (<i>int Integer double Double</i>).
Configuración:	Implementación de <i>Registrable</i> con las anotaciones <i>@New-Problem</i> y <i>@Parameters</i> , y con los parámetros recibidos en el constructor en un orden distinto al especificado.
Datos de prueba:	Objeto <i>Class<?></i> que referencia al tipo <i>Registrable</i> .
Acciones de prueba:	Pasar el objeto <i>Class</i> que hace referencia al tipo <i>Registrable</i> al método <i>validateRegistrableProblem</i> .
Resultados esperados:	Una excepción.

Test ID:	AT024
Título:	<i>Registrable</i> cuyo constructor recibe una cantidad de parámetros distintas a la esperada de acuerdo a la anotación <code>@Parameters</code> .
Característica:	Validar las anotaciones y el tipo de parámetros utilizados en el constructor de la clase <i>Registrable</i> .
Objetivo:	Validar que si el constructor público recibe más parámetros que los indicados en <code>@Parameters</code> , lanza una excepción.
Configuración:	Implementación de <i>Registrable</i> con las anotaciones <code>@New-Problem</code> y <code>@Parameters</code> , y con un parámetro extra en el constructor al indicado en la anotación <code>@Parameters</code> .
Datos de prueba:	Objeto <code>Class<?></code> que referencia al tipo <i>Registrable</i> .
Acciones de prueba:	Pasar el objeto <code>Class</code> que hace referencia al tipo <i>Registrable</i> al método <code>validateRegistrableProblem</code> .
Resultados esperados:	Una excepción.

Test ID:	AT025
Título:	<i>Registrable</i> con una anotación extra en <code>@Parameters</code> .
Característica:	Validar las anotaciones y el tipo de parámetros utilizados en el constructor de la clase <i>Registrable</i> .
Objetivo:	Validar que si <code>@Parameters</code> tiene mas anotaciones que el número de parámetros en el constructor, se lanza una excepción.
Configuración:	Implementación de <i>Registrable</i> con las anotaciones <code>@NewProblem</code> y <code>@Parameters</code> , ésta última con una anotación extra.
Datos de prueba:	Objeto <code>Class<?></code> que referencia al tipo <i>Registrable</i> .
Acciones de prueba:	Pasar el objeto <code>Class</code> que hace referencia al tipo <i>Registrable</i> al método <code>validateRegistrableProblem</code> .
Resultados esperados:	Una excepción.

Test ID:	AT026
Título:	<i>Registrable</i> con dos constructores.
Característica:	Validar las anotaciones y el tipo de parámetros utilizados en el constructor de la clase <i>Registrable</i> .
Objetivo:	Validar que si <i>Registrable</i> tiene dos constructores se lanza una excepción.
Configuración:	Implementación de <i>Registrable</i> con dos constructor, uno de ellos con las anotaciones <code>@NewProblem</code> y <code>@Parameters</code> .
Datos de prueba:	Objeto <code>Class<?></code> que referencia al tipo <i>Registrable</i> .
Acciones de prueba:	Pasar el objeto <code>Class</code> que hace referencia al tipo <i>Registrable</i> al método <code>validateRegistrableProblem</code> .
Resultados esperados:	Una excepción.

Test ID:	AT027
Título:	Id del grupo usado por <code>@NumberToggleInput</code> secuencial.
Característica:	Validar las anotaciones y el tipo de parámetros utilizados en el constructor de la clase <code>Registrable</code> .
Objetivo:	Validar que si <code>Registrable</code> , en el elemento <code>numberToggle</code> de la anotación <code>@Parameters</code> recibe las anotaciones <code>@NumberToggleInput</code> de manera secuencial no se lanza una excepción. Con secuencial se refiere a que <code>@NumberToggleInput</code> con el mismo <code>groupID</code> deben estar juntos.
Configuración:	Implementación de <code>Registrable</code> con las anotaciones <code>@NewProblem</code> y <code>@Parameters</code> . El elemento <code>numberToggle</code> de <code>@Parameters</code> tiene las anotaciones <code>@NumberToggleInput</code> con el mismo <code>groupID</code> juntos.
Datos de prueba:	Objeto <code>Class<?></code> que referencia al tipo <code>Registrable</code> .
Acciones de prueba:	Pasar el objeto <code>Class</code> que hace referencia al tipo <code>Registrable</code> al método <code>validateRegistrableProblem</code> .
Resultados esperados:	Método termina sin error.

Test ID:	AT028
Título:	Id del grupo usado por <code>@NumberToggleInput</code> no secuencial.
Característica:	Validar las anotaciones y el tipo de parámetros utilizados en el constructor de la clase <i>Registrable</i> .
Objetivo:	Validar que si <i>Registrable</i> , en el elemento <code>numberToggle</code> de la anotación <code>@Parameters</code> recibe las anotaciones <code>@NumberToggleInput</code> de manera no secuencial se lanza una excepción. Con secuencial se refiere a que <code>@NumberToggleInput</code> con el mismo <code>groupID</code> deben estar juntos.
Configuración:	Implementación de <i>Registrable</i> con las anotaciones <code>@NewProblem</code> y <code>@Parameters</code> . El elemento <code>numberToggle</code> de <code>@Parameters</code> no tiene las anotaciones <code>@NumberToggleInput</code> con el mismo <code>groupID</code> juntos.
Datos de prueba:	Objeto <code>Class<?></code> que referencia al tipo <i>Registrable</i> .
Acciones de prueba:	Pasar el objeto <code>Class</code> que hace referencia al tipo <i>Registrable</i> al método <code>validateRegistrableProblem</code> .
Resultados esperados:	Método termina sin error.

Test ID:	AT029
Título:	<i>Registrable</i> con parámetros en el constructor que no están definidos en la anotación <code>@Parameters</code> .
Característica:	Validar las anotaciones y el tipo de parámetros utilizados en el constructor de la clase <i>Registrable</i> .
Objetivo:	Validar que si el constructor <i>Registrable</i> tiene un parámetro no correspondiente al indicado en la anotación <code>@Parameters</code> se lanza una excepción.
Configuración:	Implementación de <i>Registrable</i> con las anotaciones <code>@NewProblem</code> y <code>@Parameters</code> . El constructor indica que recibirá un <i>File</i> mientras que la anotación <code>@Parameters</code> indica que en esa posición debería recibirse un <i>Object</i> , el cual hace referencia a un operador para un algoritmo metaheurístico.
Datos de prueba:	Objeto <code>Class<?></code> que referencia al tipo <i>Registrable</i> .
Acciones de prueba:	Pasar el objeto <i>Class</i> que hace referencia al tipo <i>Registrable</i> al método <code>validateRegistrableProblem</code> .
Resultados esperados:	Una excepción.

1.5. *ResultSimutation*

En esta sección se especifica los test automatizados realizados sobre la clase *ResultSimulation*.

Test ID:	AT030
Título:	<i>timeInSeconds</i> guardado correctamente.
Característica:	Validar que la clase abstracta guarda el parámetro <i>timeInSeconds</i> correctamente.
Objetivo:	Validar que al pasar el tiempo en segundos a la superclase <i>ResultSimulation</i> se guarda el tiempo correctamente en el campo <i>timeInSeconds</i> .
Configuración:	Clase que hereda de <i>ResultSimulation</i> cuyo constructor delega el parámetro <i>timeInSeconds</i> para que sea guardado en la superclase.
Datos de prueba:	<i>timeInSeconds</i> : 72000
Acciones de prueba:	Pasar al constructor de la superclase abstracta <i>ResultSimulation</i> el tiempo en segundo de la simulación.
Resultados esperados:	<i>timeInSeconds</i> igual a 72000.

Test ID:	AT031
Título:	<i>String</i> que representa el valor de <i>timeInSeconds</i> en formato HH:mm:ss.
Característica:	Validar el método <i>getTimeString</i> .
Objetivo:	Validar que al pasar el tiempo en segundos a la superclase <i>ResultSimulation</i> y llamar al método <i>getTimeString</i> se retorna un <i>String</i> con la hora en formato HH:mm:ss.
Configuración:	Clase que hereda de <i>ResultSimulation</i> cuyo constructor delega el parámetro <i>timeInSeconds</i> para que sea guardado en la superclase.
Datos de prueba:	<i>timeInSeconds</i> : 72000, 0, 1, 1000, 86159, 86399.
Acciones de prueba:	Pasar al constructor de la superclase abstracta <i>ResultSimulation</i> el tiempo en segundo de la simulación.
Resultados esperados:	Los <i>String</i> "20:00:00", "00:00:00", "00:00:01", "00:30:00", "23:55:59", "23:59:59".

Test ID:	AT032
Título:	<i>timeInSeconds</i> fuera del rango de simulación.
Característica:	Validar el intervalo permitido para <i>timeInSeconds</i>
Objetivo:	Validar que si <i>timeInSeconds</i> esta fuera del rango de las 24 horas se lanza una excepción.
Configuración:	Clase que hereda de <i>ResultSimulation</i> cuyo constructor delega el parámetro <i>timeInSeconds</i> para que sea guardado en la superclase.
Datos de prueba:	<i>timeInSeconds</i> : -2, -1, 86400(24:00:00), 86401 (24:00:01).
Acciones de prueba:	Pasar al constructor de la superclase abstracta <i>ResultSimulation</i> el tiempo en segundo de la simulación.
Resultados esperados:	Una excepción para cualquiera de los valores de <i>timeInSeconds</i> usados.

1.6. *JsonSimpleReader*

En esta sección se especifican los casos de prueba para la clase *JsonSimpleReader*.

Test ID:	AT033
Título:	Leer un entero (<i>int</i>) desde un json.
Característica:	Leer valores desde un json.
Objetivo:	Validar que el método <i>getInt</i> retorna un <i>int</i> cuando se lee un número desde una propiedad en json.
Configuración:	Instancia de <i>JsonSimpleReader</i> inicializado y listo para leer.
Datos de prueba:	<p>Un json con los siguientes valores:</p> <pre>{ "int": 5, "double": 2.5, "ints": [0,1,2,3,4,5], "doubles": [0.1,0.2,2.3,2.5,2.5], "doubleMatrix": [[0.2,0.3,0.4],[1.2,1.3,1.5]], "intMatrix": [[0,1,2],[3,4,5]], "string": "A simple string", "boolean": true }</pre>
Acciones de prueba:	Llamar al método <i>getInt</i> .
Resultados esperados:	El entero leido desde el json con la clave “int”.

Test ID:	AT034
Título:	Leer un número decimal (<i>double</i>) desde un json.
Característica:	Leer valores desde un json.
Objetivo:	Validar que el método <i>getDouble</i> retorna un <i>double</i> cuando se lee un número desde una propiedad en json.
Configuración:	Instancia de <i>JsonSimpleReader</i> inicializado y listo para leer.
Datos de prueba:	<p>Un json con los siguientes valores:</p> <pre>{ "int": 5, "double": 2.5, "ints": [0,1,2,3,4,5], "doubles": [0.1,0.2,2.3,2.5,2.5], "doubleMatrix": [[0.2,0.3,0.4],[1.2,1.3,1.5]], "intMatrix": [[0,1,2],[3,4,5]], "string": "A simple string", "boolean": true }</pre>
Acciones de prueba:	Llamar al método <i>getDouble</i> .
Resultados esperados:	El double leido desde el json con la clave “ <i>double</i> ”.

Test ID:	AT035
Título:	Leer un valor booleano (<i>boolean</i>) desde un json.
Característica:	Leer valores desde un json.
Objetivo:	Validar que el método <i>getBoolean</i> retorna un <i>boolean</i> cuando se lee una propiedad booleana desde el json.
Configuración:	Instancia de <i>JsonSimpleReader</i> inicializado y listo para leer.
Datos de prueba:	<p>Un json con los siguientes valores:</p> <pre>{ "int": 5, "double": 2.5, "ints": [0,1,2,3,4,5], "doubles": [0.1,0.2,2.3,2.5,2.5], "doubleMatrix": [[0.2,0.3,0.4],[1.2,1.3,1.5]], "intMatrix": [[0,1,2],[3,4,5]], "string": "A simple string", "boolean": true }</pre>
Acciones de prueba:	Llamar al método <i>getBoolean</i> .
Resultados esperados:	El booleano leido desde el json con la clave “ <i>boolean</i> ”.

Test ID:	AT036
Título:	Leer un arreglo de enteros (<i>int[]</i>) desde un json.
Característica:	Leer valores desde un json.
Objetivo:	Validar que el método <i>getIntegerArray</i> retorna un <i>int[]</i> cuando se lee un arreglo desde el json.
Configuración:	Instancia de <i>JsonSimpleReader</i> inicializado y listo para leer.
Datos de prueba:	<p>Un json con los siguientes valores:</p> <pre>{ "int": 5, "double": 2.5, "ints": [0,1,2,3,4,5], "doubles": [0.1,0.2,2.3,2.5,2.5], "doubleMatrix": [[0.2,0.3,0.4],[1.2,1.3,1.5]], "intMatrix": [[0,1,2],[3,4,5]], "string": "A simple string", "boolean": true }</pre>
Acciones de prueba:	Llamar al método <i>getIntegerArray</i> .
Resultados esperados:	El arreglo de enteros leidos desde el json con la clave “ints”.

Test ID:	AT037
Título:	Leer un arreglo de números decimales (<i>double[]</i>) desde un json.
Característica:	Leer valores desde un json.
Objetivo:	Validar que el método <i>getDoubleArray</i> retorna un <i>double[]</i> cuando se lee un arreglo desde el json.
Configuración:	Instancia de <i>JsonSimpleReader</i> inicializado y listo para leer.
Datos de prueba:	<p>Un json con los siguientes valores:</p> <pre>{ "int": 5, "double": 2.5, "ints": [0,1,2,3,4,5], "doubles": [0.1,0.2,2.3,2.5,2.5], "doubleMatrix": [[0.2,0.3,0.4],[1.2,1.3,1.5]], "intMatrix": [[0,1,2],[3,4,5]], "string": "A simple string", "boolean": true }</pre>
Acciones de prueba:	Llamar al método <i>getDoubleArray</i> .
Resultados esperados:	El arreglo de valores decimales leido desde el json con la clave “ <i>doubles</i> ”.

Test ID:	AT038
Título:	Leer un matriz de enteros (<i>int</i> []) desde un json.
Característica:	Leer valores desde un json.
Objetivo:	Validar que el método <i>getIntegerMatrix</i> retorna un <i>int</i> [] cuando se lee un matriz desde el json.
Configuración:	Instancia de <i>JsonSimpleReader</i> inicializado y listo para leer.
Datos de prueba:	<p>Un json con los siguientes valores:</p> <pre>{ "int": 5, "double": 2.5, "ints": [0,1,2,3,4,5], "doubles": [0.1,0.2,2.3,2.5,2.5], "doubleMatrix": [[0.2,0.3,0.4],[1.2,1.3,1.5]], "intMatrix": [[0,1,2],[3,4,5]], "string": "A simple string", "boolean": true }</pre>
Acciones de prueba:	Llamar al método <i>getIntegerMatrix</i> .
Resultados esperados:	La matriz de valores enteros leido desde el json con clave la “ <i>intMatrix</i> ”.

Test ID:	AT039
Título:	Leer un matriz de decimales (<i>double</i> [][])) desde un json.
Característica:	Leer valores desde un json.
Objetivo:	Validar que el método <i>getDoubleMatrix</i> retorna un <i>double</i> [][] cuando se lee un matriz desde el json.
Configuración:	Instancia de <i>JsonSimpleReader</i> inicializado y listo para leer.
Datos de prueba:	<p>Un json con los siguientes valores:</p> <pre>{ "int": 5, "double": 2.5, "ints": [0,1,2,3,4,5], "doubles": [0.1,0.2,2.3,2.5,2.5], "doubleMatrix": [[0.2,0.3,0.4],[1.2,1.3,1.5]], "intMatrix": [[0,1,2],[3,4,5]], "string": "A simple string", "boolean": true }</pre>
Acciones de prueba:	Llamar al método <i>getDoubleMatrix</i> .
Resultados esperados:	La matriz de valores decimales leido desde el json con la clave “ <i>doubleMatrix</i> ”.

2. Casos de pruebas manuales

En este capítulo se presenta la especificación formal de los casos de prueba manuales que son realizados en el software. Estas pruebas son aquellas que se realizan con la ayuda de interacción humana.

Test ID:	MT001
Título:	Visualización de la red.
Característica:	Mostrar visualización de la red.
Objetivo:	Confirmar que la red puede ser leída desde un archivo con extensión “.inp” y ser visualizada en la aplicación.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	inp: “hanoi-Frankenstein.inp” .
Acciones de prueba:	1. Abrir JHawanetFramework. 2. Cargar archivo de red.
Resultados esperados:	El sistema muestra la red leída desde un archivo inp gráficamente en la aplicación.

Test ID:	MT002
Título:	Optimización monoobjetivo realizada completamente.
Característica:	Realizar simulación monoobjetivo.
Objetivo:	Confirmar que se puede llevar a cabo la resolución del problema <i>Pipe Optimizing</i> sobre la red abierta.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	<p><i>independentRun</i> = 10.</p> <p>Archivo inp: “hanoi-Frankenstein.inp”.</p> <p>Archivo gama: “hanoiHW.Gama”.</p>
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Seleccionar el problema <i>Pipe Optimizing</i> del menú. 4. Configurar el problema usando la ventana de configuración. 5. Realizar la optimización.
Resultados esperados:	Al terminar la optimización el sistema muestra una interfaz con las soluciones generadas por la optimización. Debe haber tantas soluciones como el numero de configuraciones independientes (<i>independentRun</i>) establecidas.

Test ID:	MT003
Título:	Optimización monoobjetivo cancelada.
Característica:	Realizar simulación monoobjetivo.
Objetivo:	Confirmar que se puede llevar a cabo la resolución del problema <i>Pipe Optimizing</i> sobre la red abierta y que se puede cancelar el proceso cerrando la ventana o pulsando cancelar.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “hanoi-Frankenstein.inp”. Archivo gama: “hanoiHW.Gama”.
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Seleccionar el problema <i>Pipe Optimizing</i> del menú. 4. Configurar el problema usando la ventana de configuración. 5. Realizar la optimización.
Resultados esperados:	Al cancelar la búsqueda de soluciones la ventana de estado indica que la optimización ha sido detenida.

Test ID:	MT004
Título:	Optimización multiobjetivo realizada completamente.
Característica:	Realizar simulación multiobjetivo.
Objetivo:	Confirmar que se puede llevar a cabo la resolución del problema <i>Pumping Schedule</i> sobre la red abierta.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “Vanzyl.inp”. Archivo gama: “VanzylConfiguration.json”.
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Seleccionar el problema <i>Pumping Schedule</i> del menú. 4. Seleccionar el algoritmo NSGAII. 5. Configurar el problema usando la ventana de configuración. 6. Realizar la optimización.
Resultados esperados:	Al terminar la optimización el sistema muestra una interfaz con las soluciones generadas por la optimización. Estas soluciones corresponden a la Frontera de Pareto del Experimento.

Test ID:	MT005
Título:	Optimización multiobjetivo cancelada.
Característica:	Realizar simulación multiobjetivo.
Objetivo:	Confirmar que se puede llevar a cabo la resolución del problema <i>Pumping Schedule</i> sobre la red abierta y que se puede cancelar el proceso cerrando la ventana o pulsando cancelar.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “Vanzyl.inp”. Archivo gama: “VanzylConfiguration.json”.
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Seleccionar el problema <i>Pumping Schedule</i> del menú. 4. Seleccionar el algoritmo NSGAII. 5. Configurar el problema usando la ventana de configuración. 6. Realizar la optimización.
Resultados esperados:	Al cancelar la búsqueda de soluciones la ventana de estado indica que la optimización ha sido detenida.

Test ID:	MT006
Título:	Ver soluciones de los problemas monoobjetivos gráficamente a medida que se ejecuta la optimización.
Característica:	Visualizar gráficamente las soluciones.
Objetivo:	Comprobar que a medida que algoritmo va generando soluciones éstas pueden ser visualizadas. El gráfico es Objetivo vs Numero de generaciones.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “hanoi-Frankenstein.inp”. Archivo gama: “hanoiHW.Gama”.
Acciones de prueba:	1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Escoger el problema monoobjetivo. 4. Configurar el problema monoobjetivo y ejecutar. 5. Ir a la ventana del gráfico.
Resultados esperados:	Un gráfico de dos dimensiones en el que se muestre los resultados de las soluciones generadas por cada generación de cada una de las ejecuciones independientes.

Test ID:	MT007
Título:	Ver soluciones del problema multiobjetivos, con dos objetivos, gráficamente a medida que se ejecuta la optimización.
Característica:	Visualizar gráficamente las soluciones
Objetivo:	Comprobar que a medida que algoritmo va generando soluciones estas pueden ser visualizadas. El gráfico es Objetivo1 vs Objetivo2.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “Vanzyl.inp”. Archivo gama: “VanzylConfiguration.json”.
Acciones de prueba:	1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Escoger el problema monoobjetivo. 4. Configurar el problema monoobjetivo y ejecutar. 5. Ir a la ventana del gráfico.
Resultados esperados:	Un gráfico de dos dimensiones en el que se muestre los resultados de las soluciones generadas por cada generación de cada una de las ejecuciones independientes.

Test ID:	MT008
Título:	Guardar la gráfica de las soluciones del problema monoobjetivo como png.
Característica:	Guardar gráfica de soluciones.
Objetivo:	Confirmar que se puede guardar el gráfico de soluciones como un archivo png.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “hanoi-Frankenstein.inp”. Archivo gama: “hanoiHW.Gama”.
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Escoger el problema monoobjetivo. 4. Configurar el problema monoobjetivo y ejecutar. 5. Ir a la ventana del gráfico. 6. Pulsar el botón para guardar una captura del gráfico. 7. Configurar donde guardar la captura.
Resultados esperados:	Generación de un archivo png en el equipo.

Test ID:	MT009
Título:	Guardar la gráfica de soluciones de problemas de 2 objetivos como png.
Característica:	Guardar gráfica de soluciones.
Objetivo:	Confirmar que se puede guardar el gráfico de soluciones como un png.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “Vanzyl.inp”. Archivo gama: “VanzylConfiguration.json”.
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Escoger el problema monoobjetivo. 4. Configurar el problema monoobjetivo y ejecutar. 5. Ir a la ventana del gráfico. 6. Pulsar el botón para guardar una captura del gráfico. 7. Configurar donde guardar la captura.
Resultados esperados:	Generación de un archivo png en el equipo.

Test ID:	MT010
Título:	Guardar la solución seleccionada como inp.
Característica:	Guardar resultados de la optimización.
Objetivo:	Confirmar que se puede exportar los resultados de la solución sobre el archivo de red (inp).
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “hanoi-Frankenstein.inp” Archivo gama: “hanoiHW.Gama”
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Escoger el problema monoobjetivo. 4. Configurar el problema monoobjetivo y ejecutar. 5. Esperar que la ejecución finalice. 6. Seleccionar una solución. 7. Pulsar el botón guardar como inp. 8. Configurar donde guardar el archivo.
Resultados esperados:	Generación del archivo de configuración de la red (inp) con la solución aplicada.

Test ID:	MT011
Título:	Guardar las soluciones en archivos tsv.
Característica:	Guardar resultados de la optimización.
Objetivo:	Confirmar que se puede exportar las soluciones a dos archivos tsv. Uno para las variables y el otro para los objetivos.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “hanoi-Frankenstein.inp”. Archivo gama: “hanoiHW.Gama”.
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Escoger el problema monoobjetivo. 4. Configurar el problema monoobjetivo y ejecutar. 5. Esperar que la ejecución finalice. 6. Pulsar el botón guardar tabla. 7. Configurar donde guardar los archivos.
Resultados esperados:	Generación de los 2 archivos .tsv. Uno con el prefijo FUN_ y otro con el prefijo VAR_. El archivo FUN contiene el valor de los objetivos de las soluciones. El archivo VAR contiene las variables de las soluciones.

Test ID:	MT012
Título:	Guardar las soluciones en un Excel.
Característica:	Guardar resultados de la optimización.
Objetivo:	Confirmar que se puede exportar la tabla de resultados completa a un Excel.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “hanoi-Frankenstein.inp”. Archivo gama: “hanoiHW.Gama”.
Acciones de prueba:	<ol style="list-style-type: none"> 1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Escoger el problema monoobjetivo. 4. Configurar el problema monoobjetivo y ejecutar. 5. Esperar que la ejecución finalice. 6. Pulsar el botón guardar tabla como Excel. 7. Configurar donde guardar el archivo.
Resultados esperados:	Un archivo Excel con los mismos datos que la tabla de resultados de la aplicación.

Test ID:	MT013
Título:	Simulación hidráulica de la red en un solo período de tiempo.
Característica:	Realizar simulación hidráulica con los valores establecidos en el archivo de configuración de red.
Objetivo:	Confirmar que se puede realizar la simulación utilizando los valores del archivo de configuración de red.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “hanoi-Frankenstein.inp”. Archivo gama: “hanoiHW.Gama”.
Acciones de prueba:	1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Pulsar botón <i>Execute</i> de la ventana principal.
Resultados esperados:	Ejecución realizada sin ningun error.

Test ID:	MT014
Título:	Ver resultados de la simulación hidráulica de un solo período de tiempo
Característica:	Realizar simulación hidráulica con los valores establecidos en el archivo de configuración de red.
Objetivo:	Confirmar que se puede visualizar los resultados de la simulación realizada.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “hanoi-Frankenstein.inp”. Archivo gama: “hanoiHW.Gama”.
Acciones de prueba:	1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Pulsar botón <i>Execute</i> de la ventana principal. 4. Pulsar botón <i>Results</i> de la ventana principal.
Resultados esperados:	Una interfaz que permite seleccionar si se quiere ver los resultados para los enlaces o los nodos.

Test ID:	MT015
Título:	Simulación hidráulica de la red de más de un período de tiempo de simulación.
Característica:	Realizar simulación hidráulica con los valores establecidos en el archivo de configuración de red.
Objetivo:	Confirmar que se puede realizar la simulación utilizando los valores del archivo de configuración de red.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “Vanzyl.inp”. Archivo gama: “VanzylConfiguration.json”.
Acciones de prueba:	1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Pulsar botón <i>Execute</i> de la ventana principal.
Resultados esperados:	Ejecución realizada sin ningún error.

Test ID:	MT016
Título:	Ver resultados de la simulación hidráulica de mas de un período de tiempo
Característica:	Realizar simulación hidráulica con los valores establecidos en el archivo de configuración de red.
Objetivo:	Confirmar que se puede visualizar los resultados de la simulación realizada.
Configuración:	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
Datos de prueba:	Archivo inp: “Vanzyl.inp”. Archivo gama: “VanzylConfiguration.json”.
Acciones de prueba:	1. Abrir JHawanetFramework. 2. Cargar archivo de red. 3. Pulsar botón <i>Execute</i> de la ventana principal. 4. Pulsar botón <i>Results</i> de la ventana principal.
Resultados esperados:	Una interfaz que permite seleccionar si se quieren ver los resultados para los enlaces o los nodos. Adicionalmente, permite escoger el tiempo de simulación y listar los resultados para todos los tiempos de un elemento de la red específico.

E. Encuesta para la evaluación de la aplicación

Formulario de evaluación de la aplicación JHawanetFramework

Este formulario tiene como fin el evaluar la funcionalidad, usabilidad y utilidad del software desarrollado.

Elija que grado de cumplimiento o satisfacción tiene con respecto a los siguientes enunciados
***Obligatorio**

1. Por favor, indique su nombre y apellido. *

Funcionalidad

Marque solo un óvalo por fila.

2. *

Marca solo un óvalo por fila.

	Totalmente en desacuerdo	En desacuerdo	Ni en acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
La aplicación permite visualizar la red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
La aplicación permite visualizar la configuración de los elementos de la red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
La aplicación permite realizar la simulación hidráulica sobre una red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
La aplicación permite resolver desde el enfoque monoobjetivo problemas asociados a las RDA.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
La aplicación permite resolver desde el enfoque multiobjetivo problemas asociados a las RDA.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
La aplicación permite visualizar un gráfico para mostrar las soluciones del problema monoobjetivo.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
La aplicación permite visualizar un	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

gráfico para mostrar las soluciones del problema multiobjetivo.

La aplicación permite exportar el gráfico a una imagen.

La aplicación muestra los resultados obtenidos de los problemas monoobjetivo.

La aplicación muestra los resultados obtenidos de los problemas multiobjetivo.

La aplicación permite exportar la tabla de resultados de los problemas a Excel.

La aplicación permite exportar únicamente los objetivos y variables a archivos tsv.

La aplicación permite seleccionar una solución para modificar la red original.

Facilidad de uso

Marque solo un ovalo por fila.

3. *

Marca solo un óvalo por fila.

	Totalmente en desacuerdo	En desacuerdo	Ni en acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
Creo que usaría esta aplicación frecuentemente.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Encuentro esta aplicación innecesariamente complejo.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Creo que la aplicación fue fácil de usar.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Creo que necesitaría ayuda de una persona con conocimientos técnicos para usar esta aplicación.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Las funciones de esta aplicación están bien integradas.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Creo que la aplicación es muy inconsistente.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Imagino que la mayoría de la gente aprendería a usar esta aplicación en forma muy rápida.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Encuentro que la aplicación es muy difícil de usar.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Me siento confiado al usar esta	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

aplicación.

Necesité aprender
muchas cosas antes
de ser capaz de usar
esta aplicación.

Utilidad

Marque solo un ovalo por fila.

4. *

Marca solo un óvalo por fila.

	Totalmente en desacuerdo	En desacuerdo	Ni en acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
Es útil para usted poder realizar la simulación hidráulica de la red por defecto desde la misma aplicación.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Los resultados mostrados cuando se realiza una simulación hidráulica utilizando la configuración de la red por defecto son adecuados.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
El gráfico que se muestra al resolver problemas monoobjetivos es adecuado.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
El gráfico que se muestra al resolver problemas multiobjetivos es adecuado.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Los atributos mostrados en la tabla de resultados de los problemas monoobjetivos son los adecuados.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Los atributos mostrados en la tabla de resultados de los problemas multiobjetivo son los adecuados.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Es útil para usted poder exportar la tabla de resultados de los problemas a Excel.

Es útil para usted poder seleccionar una solución y generar un nuevo archivo de red de acuerdo a esta.

Es útil para usted poder guardar los valores de las soluciones en dos archivos diferentes. Uno con los objetivos y el otro con las variables.

Preguntas

En esta sección se solicita responder las siguientes preguntas.

5. ¿El sistema presento algún error durante su uso?

6. ¿Qué cambios o mejoras le haría usted a la aplicación?

Formulario de evaluación del manual de usuario de la aplicación.

Utilidad

Marque solo un óvalo por fila.

7. *

Marca solo un óvalo por fila.

	Totalmente en desacuerdo	En desacuerdo	Ni en acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
Fue útil para usted contar con un manual de usuario para agregar nuevos algoritmos, problemas y operadores.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Los temas explicados en el manual de usuario fueron los apropiados.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Fue el lenguaje usado para escribir el manual usuario adecuado.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

8. ¿Encontró alguna sección del manual de usuario difícil de entender o que debiera ser mejorada?

Por favor indique cual sección y que error encontró.

9. ¿Hay algún tema de la aplicación que crea que debe ser incluido en el manual de usuario?

Por favor indique que temas debieran ser agregados al manual de usuario con del fin de facilitar el uso de la aplicación.

Este contenido no ha sido creado ni aprobado por Google.

Google Formularios

F. Respuestas de la encuesta aplicada para evaluar la aplicación

Formulario de evaluación de la aplicación JHawanetFramework

Este formulario tiene como fin el evaluar la funcionalidad, usabilidad y utilidad del software desarrollado.

Elija que grado de cumplimiento o satisfacción tiene con respecto a los siguientes enunciados

Por favor, indique su nombre y apellido. *

Yamisleydi Salgueiro

Funcionalidad

Marque solo un óvalo por fila.

*

	Totalmente en desacuerdo	En desacuerdo	Ni en acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
La aplicación permite visualizar la red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite visualizar la configuración de los elementos de la red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite realizar la simulación hidráulica sobre una red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite resolver desde el enfoque monoobjetivo problemas asociados a las RDA.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite resolver desde el enfoque multiobjetivo problemas asociados a las RDA.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite visualizar un gráfico para mostrar las soluciones del problema monoobjetivo.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

La aplicación permite visualizar un gráfico para mostrar las soluciones del problema multiobjetivo.

La aplicación permite exportar el gráfico a una imagen.

La aplicación muestra los resultados obtenidos de los problemas monoobjetivo.

La aplicación muestra los resultados obtenidos de los problemas multiobjetivo.

La aplicación permite exportar la tabla de resultados de los problemas a Excel.

La aplicación permite exportar únicamente los objetivos y variables a archivos tsv.

La aplicación permite seleccionar una solución para modificar la red original.

Facilidad de uso

Marque solo un ovalo por fila.

*

	Totalmente en desacuerdo	En desacuerdo	Ni en acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
Creo que usaría esta aplicación frecuentemente.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Encuentro esta aplicación innecesariamente complejo.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Creo que la aplicación fue fácil de usar.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Creo que necesitaría ayuda de una persona con conocimientos técnicos para usar esta aplicación.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Las funciones de esta aplicación están bien integradas.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Creo que la aplicación es muy inconsistente.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Imagino que la mayoría de la gente aprendería a usar esta aplicación en forma muy rápida.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Encuentro que la aplicación es muy difícil de usar.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>

Me siento
confiado al usar
esta aplicación.

Necesité aprender
muchas cosas
antes de ser
capaz de usar
esta aplicación.



Utilidad

Marque solo un ovalo por fila.

*

Totalmente en desacuerdo En desacuerdo Ni en acuerdo ni en desacuerdo De acuerdo Totalmente de acuerdo

Es útil para usted poder realizar la simulación hidráulica de la red por defecto desde la misma aplicación.

Los resultados mostrados cuando se realiza una simulación hidráulica utilizando la configuración de la red por defecto son adecuados.

El gráfico que se muestra al resolver problemas monoobjetivos es adecuado.

El gráfico que se muestra al resolver problemas multiobjetivos es adecuado.

Los atributos mostrados en la tabla de resultados de los problemas monoobjetivos son los adecuados.

Los atributos mostrados en la tabla de resultados de los problemas multiobjetivo son los adecuados.

Es útil para usted poder exportar la tabla de resultados de los problemas a Excel.

Es útil para usted poder seleccionar una solución y generar un nuevo archivo de red de acuerdo a esta.

Es útil para usted poder guardar los valores de las soluciones en dos archivos diferentes. Uno con los objetivos y el otro con las variables.

Preguntas

En esta sección se solicita responder las siguientes preguntas.

¿El sistema presento algún error durante su uso?

No

¿Qué cambios o mejoras le haría usted a la aplicación?

Mejorar la ayuda, de esta forma cualquier usuario podría utilizar la herramienta.

Formulario de evaluación del manual de usuario de la aplicación.

Utilidad

Marque solo un ovalo por fila.

*

	Totalmente en desacuerdo	En desacuerdo	Ni en acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
--	--------------------------	---------------	--------------------------------	------------	-----------------------

Fue útil para usted contar con un manual de usuario para agregar nuevos algoritmos, problemas y operadores.

<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
-----------------------	-----------------------	----------------------------------	-----------------------	-----------------------

Los temas explicados en el manual de usuario fueron los apropiados.

<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
-----------------------	-----------------------	----------------------------------	-----------------------	-----------------------

Fue el lenguaje usado para escribir el manual usuario adecuado.

<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
-----------------------	-----------------------	----------------------------------	-----------------------	-----------------------

¿Encontró alguna sección del manual de usuario difícil de entender o que debiera ser mejorada?

Por favor indique cual sección y que error encontró.

No encontré el manual de usuario entre los archivo enviados. Además, no pude abrir la ayuda dentro del sistema.

¿Hay algún tema de la aplicación que crea que debe ser incluido en el manual de usuario?

Por favor indique que temas debieran ser agregados al manual de usuario con del fin de facilitar el uso de la aplicación.

No

Este contenido no ha sido creado ni aprobado por Google.

Google Formularios

Formulario de evaluación de la aplicación JHawanetFramework

Este formulario tiene como fin el evaluar la funcionalidad, usabilidad y utilidad del software desarrollado.

Elija que grado de cumplimiento o satisfacción tiene con respecto a los siguientes enunciados

Por favor, indique su nombre y apellido. *

Marco Alsina

Funcionalidad

Marque solo un ovalo por fila.

*

	Totalmente en desacuerdo	En desacuerdo	Ni en acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
La aplicación permite visualizar la red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite visualizar la configuración de los elementos de la red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite realizar la simulación hidráulica sobre una red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
La aplicación permite resolver desde el enfoque monoobjetivo problemas asociados a las RDA.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
La aplicación permite resolver desde el enfoque multiobjetivo problemas asociados a las RDA.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
La aplicación permite visualizar un gráfico para mostrar las soluciones del problema monoobjetivo.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

La aplicación permite visualizar un gráfico para mostrar las soluciones del problema multiobjetivo.

La aplicación permite exportar el gráfico a una imagen.

La aplicación muestra los resultados obtenidos de los problemas monoobjetivo.

La aplicación muestra los resultados obtenidos de los problemas multiobjetivo.

La aplicación permite exportar la tabla de resultados de los problemas a Excel.

La aplicación permite exportar únicamente los objetivos y variables a archivos tsv.

La aplicación permite seleccionar una solución para modificar la red original.

Facilidad de uso

Marque solo un ovalo por fila.

*

	Totalmente en desacuerdo	En desacuerdo	Ni en acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
Creo que usaría esta aplicación frecuentemente.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Encuentro esta aplicación innecesariamente complejo.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Creo que la aplicación fue fácil de usar.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Creo que necesitaría ayuda de una persona con conocimientos técnicos para usar esta aplicación.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Las funciones de esta aplicación están bien integradas.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Creo que la aplicación es muy inconsistente.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Imagino que la mayoría de la gente aprendería a usar esta aplicación en forma muy rápida.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Encuentro que la aplicación es muy difícil de usar.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Me siento
confiado al usar
esta aplicación.

Necesité aprender
muchas cosas
antes de ser
capaz de usar
esta aplicación.

Utilidad

Marque solo un ovalo por fila.

*

Totalmente en desacuerdo En desacuerdo Ni en acuerdo ni en desacuerdo De acuerdo Totalmente de acuerdo

Es útil para usted poder realizar la simulación hidráulica de la red por defecto desde la misma aplicación.

Los resultados mostrados cuando se realiza una simulación hidráulica utilizando la configuración de la red por defecto son adecuados.

El gráfico que se muestra al resolver problemas monoobjetivos es adecuado.

El gráfico que se muestra al resolver problemas multiobjetivos es adecuado.

Los atributos mostrados en la tabla de resultados de los problemas monoobjetivos son los adecuados.

Los atributos mostrados en la tabla de resultados de los problemas multiobjetivo son los adecuados.

Es útil para usted poder exportar la tabla de resultados de los problemas a Excel.

Es útil para usted poder seleccionar una solución y generar un nuevo archivo de red de acuerdo a esta.

Es útil para usted poder guardar los valores de las soluciones en dos archivos diferentes. Uno con los objetivos y el otro con las variables.

Preguntas

En esta sección se solicita responder las siguientes preguntas.

¿El sistema presento algún error durante su uso?

Si. Los errores se asociaron a la ausencia de Epanet en mi equipo (resuelto), y al parecer una versión incorrecta de JRE que restaba de toda funcionalidad al programa.

¿Qué cambios o mejoras le haría usted a la aplicación?

Agregaría un manual de usuario, que explicara con suficiente detalle que es lo que hace cada opción del programa, ideal con ejemplos guiados (e.g. como resolver la red, como modificar la red, como correr una optimización), y detallando los valores de columna presentados en la tabla de resultados.

Formulario de evaluación del manual de usuario de la aplicación.

Utilidad

Marque solo un ovalo por fila.

*

Totalmente en desacuerdo En desacuerdo Ni en acuerdo ni en desacuerdo De acuerdo Totalmente de acuerdo

Fue útil para usted contar con un manual de usuario para agregar nuevos algoritmos, problemas y operadores.

Los temas explicados en el manual de usuario fueron los apropiados.

Fue el lenguaje usado para escribir el manual usuario adecuado.

¿Encontró alguna sección del manual de usuario difícil de entender o que debiera ser mejorada?

Por favor indique cual sección y que error encontró.

El manual de usuario actual del framework no indica el propósito del programa, no otorga un contexto para determinar si es relevante su uso para el lector, ni indica el público objetivo que podría estar interesado en utilizar el framework.

¿Hay algún tema de la aplicación que crea que debe ser incluido en el manual de usuario?

Por favor indique que temas debieran ser agregados al manual de usuario con el fin de facilitar el uso de la aplicación.

Es necesario un manual de usuario del programa mismo, mas allá del framework

Este contenido no ha sido creado ni aprobado por Google.

Google Formularios

Formulario de evaluación de la aplicación JHawanetFramework

Este formulario tiene como fin el evaluar la funcionalidad, usabilidad y utilidad del software desarrollado.

Elija que grado de cumplimiento o satisfacción tiene con respecto a los siguientes enunciados

Por favor, indique su nombre y apellido. *

Sergio Silva Rubio

Funcionalidad

Marque solo un ovalo por fila.

*

	Totalmente en desacuerdo	En desacuerdo	Ni en acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
La aplicación permite visualizar la red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite visualizar la configuración de los elementos de la red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite realizar la simulación hidráulica sobre una red.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite resolver desde el enfoque monoobjetivo problemas asociados a las RDA.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite resolver desde el enfoque multiobjetivo problemas asociados a las RDA.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
La aplicación permite visualizar un gráfico para mostrar las soluciones del problema monoobjetivo.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>

La aplicación permite visualizar un gráfico para mostrar las soluciones del problema multiobjetivo.

La aplicación permite exportar el gráfico a una imagen.

La aplicación muestra los resultados obtenidos de los problemas monoobjetivo.

La aplicación muestra los resultados obtenidos de los problemas multiobjetivo.

La aplicación permite exportar la tabla de resultados de los problemas a Excel.

La aplicación permite exportar únicamente los objetivos y variables a archivos tsv.

La aplicación permite seleccionar una solución para modificar la red original.

Facilidad de uso

Marque solo un ovalo por fila.

*

	Totalmente en desacuerdo	En desacuerdo	Ni en acuerdo ni en desacuerdo	De acuerdo	Totalmente de acuerdo
Creo que usaría esta aplicación frecuentemente.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Encuentro esta aplicación innecesariamente complejo.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Creo que la aplicación fue fácil de usar.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Creo que necesitaría ayuda de una persona con conocimientos técnicos para usar esta aplicación.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Las funciones de esta aplicación están bien integradas.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Creo que la aplicación es muy inconsistente.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Imagino que la mayoría de la gente aprendería a usar esta aplicación en forma muy rápida.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Encuentro que la aplicación es muy difícil de usar.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Me siento
confiado al usar
esta aplicación.

Necesité aprender
muchas cosas
antes de ser
capaz de usar
esta aplicación.



Utilidad

Marque solo un ovalo por fila.

*

Totalmente en desacuerdo En desacuerdo Ni en acuerdo ni en desacuerdo De acuerdo Totalmente de acuerdo

Es útil para usted poder realizar la simulación hidráulica de la red por defecto desde la misma aplicación.

Los resultados mostrados cuando se realiza una simulación hidráulica utilizando la configuración de la red por defecto son adecuados.

El gráfico que se muestra al resolver problemas monoobjetivos es adecuado.

El gráfico que se muestra al resolver problemas multiobjetivos es adecuado.

Los atributos mostrados en la tabla de resultados de los problemas monoobjetivos son los adecuados.

Los atributos mostrados en la tabla de resultados de los problemas multiobjetivo son los adecuados.

Es útil para usted poder exportar la tabla de resultados de los problemas a Excel.

Es útil para usted poder seleccionar una solución y generar un nuevo archivo de red de acuerdo a esta.

Es útil para usted poder guardar los valores de las soluciones en dos archivos diferentes. Uno con los objetivos y el otro con las variables.

Preguntas

En esta sección se solicita responder las siguientes preguntas.

¿El sistema presento algún error durante su uso?

Error no. Solo No me fué fácil encontrar los archivos "gama" y "json".

¿Qué cambios o mejoras le haría usted a la aplicación?

Redondear números en algún decimal. Validar datos en formulario. Mostrar las soluciones de un problema multiobjetivo, no solo su representación entera.

Formulario de evaluación del manual de usuario de la aplicación.

Utilidad

Marque solo un ovalo por fila.

*

Totalmente en desacuerdo En desacuerdo Ni en acuerdo ni en desacuerdo De acuerdo Totalmente de acuerdo

Fue útil para usted contar con un manual de usuario para agregar nuevos algoritmos, problemas y operadores.

Los temas explicados en el manual de usuario fueron los apropiados.

Fue el lenguaje usado para escribir el manual usuario adecuado.

¿Encontró alguna sección del manual de usuario difícil de entender o que debiera ser mejorada?

Por favor indique cual sección y que error encontró.

Creo que falta una introducción explicando algunos aspectos como: en qué consiste el manual, por qué es necesario, sobre qué lenguaje se está trabajando, agregar algunos detalles de para qué y por qué se utiliza @notaciones por ejemplo, etc.

¿Hay algún tema de la aplicación que crea que debe ser incluido en el manual de usuario?

Por favor indique que temas debieran ser agregados al manual de usuario con el fin de facilitar el uso de la aplicación.

Un ejemplo base sobre el uso de la aplicación y que permita probar que este todo funcionando. También se podría agregar una sección de "troubleshooting" con solución a errores típicos

Este contenido no ha sido creado ni aprobado por Google.

Google Formularios