



**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  
08, 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

08, 2020

*Dedicado a ...*

## AGRADECIMIENTOS

Agradecimientos a ...

## TABLA DE CONTENIDOS

	página
<b>Dedicatoria</b>	<b>I</b>
<b>Agradecimientos</b>	<b>II</b>
<b>Tabla de Contenidos</b>	<b>III</b>
<b>Índice de Figuras</b>	<b>VI</b>
<b>Índice de Tablas</b>	<b>VIII</b>
<b>Resumen</b>	<b>IX</b>
<b>1. Introducción</b>	<b>10</b>
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
<b>2. Marco Teórico</b>	<b>15</b>
2.1. Metodología de desarrollo . . . . .	15
2.2. Metodología de evaluación . . . . .	16
2.3. Herramientas para la implementación del software . . . . .	17
2.3.1. Java . . . . .	17
2.3.2. Inversión de control . . . . .	19
2.3.3. Inyección de dependencias . . . . .	20
2.3.4. Epanet Programming Toolkit . . . . .	20
2.4. Redes de agua potable . . . . .	20
2.4.1. Red de distribución de agua . . . . .	21
2.5. Optimización . . . . .	22
2.6. Heurística . . . . .	23
2.7. Metaheurística . . . . .	23

2.7.1.	Algoritmos Evolutivos . . . . .	24
2.7.2.	Algoritmo Genético . . . . .	25
2.7.3.	Conceptos para la optimización multiobjetivo . . . . .	26
2.7.4.	Algoritmo NSGAI ( <i>Non-Dominated Sorting Genetic Algorithm II</i> ) . . . . .	27
2.8.	Trabajo relacionado . . . . .	32
<b>3.</b>	<b>Metodología de desarrollo</b>	<b>36</b>
<b>4.</b>	<b>Desarrollo</b>	<b>40</b>
4.1.	Concepción del proyecto . . . . .	40
4.2.	Casos de prueba . . . . .	40
4.3.	Planificación . . . . .	41
4.4.	Requisitos . . . . .	42
4.5.	Diseño . . . . .	45
4.5.1.	Detalles de implementación . . . . .	50
4.6.	Implementación . . . . .	62
4.6.1.	Modelo matemático de los problemas . . . . .	71
4.7.	Pruebas . . . . .	75
<b>5.</b>	<b>Evaluación de la solución</b>	<b>84</b>
5.1.	Metodología de evaluación . . . . .	84
5.2.	Diseño del estudio de caso . . . . .	84
5.2.1.	Elección del caso . . . . .	85
5.2.2.	Objetivos de la investigación . . . . .	85
5.2.3.	Características a evaluar . . . . .	85
5.2.4.	Protocolo para conducir el estudio de caso . . . . .	86
5.2.5.	Unidad de análisis . . . . .	86
5.3.	Consideraciones preliminar . . . . .	86
5.4.	Recolección de datos . . . . .	87
5.5.	Análisis de datos . . . . .	88
5.5.1.	Funcionalidad . . . . .	89
5.5.2.	Usabilidad . . . . .	90
5.5.3.	Utilidad . . . . .	90
5.5.4.	Utilidad del manual de usuario . . . . .	92

5.6. Conclusiones del estudio . . . . .	93
<b>Glosario</b>	<b>94</b>
<b>Bibliografía</b>	<b>95</b>
<b>Anexos</b>	
<b>A: Documento de especificación de requisitos</b>	<b>100</b>
<b>B: Documento de diseño</b>	<b>101</b>
<b>C: Manual de usuario</b>	<b>102</b>
<b>D: Documento de casos de prueba</b>	<b>103</b>
<b>E: Cuestionario para la evaluación de la aplicación</b>	<b>104</b>

## 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. Pseudocódigo Algoritmo Evolutivo . . . . .	24
2.4. Ejemplo de dominancia y Óptimo de Pareto . . . . .	27
2.5. Ejemplo frente de Pareto . . . . .	28
2.6. Pseudocódigo de la función de remplazo utilizada en el algoritmo NS- GAII . . . . .	29
2.7. Procedimiento NSGAII . . . . .	30
2.8. Frentes no dominados . . . . .	31
2.9. Pseudocódigo de la función de ordenamiento utilizada en NSGAII . .	34
2.10. Cálculo de la densidad de estimación al rededor de la solución $i$ . . .	35
2.11. Pseudocódigo de la función de asignación de densidad . . . . .	35
4.1. Arquitectura física monolítica . . . . .	47
4.2. Arquitectura lógica Modelo-Vista-Controlador . . . . .	48
4.3. Diagrama de clases de la abstracción de la red reducido. . . . .	53
4.4. Diagrama de secuencia de la carga y visualización de la red . . . . .	54
4.5. Diagrama de clases modulo metaheurística. Modificación a partir del diagrama presentado en [9] . . . . .	55
4.6. Código del método <i>runSingleStep</i> utilizado con GA y NSGAII . . . .	56
4.7. Diagrama de secuencia para llevar a cabo la optimizacion de los pro- blemas . . . . .	57
4.8. Diagrama de clases para la simulación hidráulica . . . . .	58
4.9. Diagrama de secuencia para realizar la simulación hidráulica . . . . .	59
4.10. Diagrama de actividades de la aplicación. Durante “Configurar Expe- rimento” se leen las anotaciones para construir la interfaz gráfica, una vez configurado el experimento se crean e inyectan las dependencias. .	60
4.11. Interfaz de clase <i>Registrable</i> . . . . .	61
4.12. Ventana principal de la aplicación donde se visualiza la red . . . . .	64
4.13. Ventana de descripción del problema. . . . .	65
4.14. Ventana de configuración del problema. . . . .	66



4.15. Ventana de configuración de parámetros del operador <i>UniformSelection</i>	66
4.16. Ventana del retroalimentación mostrada durante la ejecución . . . . .	67
4.17. Ventana de resultados generada cuando termina la ejecución para el problema monoobjetivo <i>Pipe Optimizing</i> . . . . .	68
4.18. Ventana de simulación hidráulica utilizando los valores del archivo de red . . . . .	69
4.19. Uso de anotaciones en la clase que hereda <i>Registrable</i> para construir la interfaz de configuración del problema . . . . .	70
4.20. Representación de la solución del problema monoobjetivo <i>Pipe Opti- mizing</i> . . . . .	72
4.21. Representación de la solución problema multiobjetivo <i>Pumping Schedule</i>	75
5.1. Gráfico circular de los resultados de la evaluación de funcionalidad de la aplicación . . . . .	89
5.2. Gráfico circular de los resultados de la evaluación de utilidad de la aplicación . . . . .	91
5.3. Gráfico circular de los resultados de la evaluación de la utilidad del manual de usuario . . . . .	92

## ÍNDICE DE TABLAS

	página
4.1. Planificación de las iteraciones . . . . .	42
4.2. Actividades y cambios de la fase de requisitos durante cada iteración	43
4.3. Especificación del requisito de usuario RU004. . . . .	44
4.4. Especificación del requisito de usuario RU002. . . . .	44
4.5. Especificación del requisito de usuario RU020. . . . .	45
4.6. Actividades y cambios en la fase de diseño durante cada iteración . .	46
4.7. Actividades fase de implementación . . . . .	62
4.8. Especificación caso de prueba manual MT001 . . . . .	77
4.9. Especificación caso de prueba manual MT002 . . . . .	78
4.10. Especificación caso de prueba manual MT003 . . . . .	79
4.11. Especificación caso de prueba manual MT013 . . . . .	80
4.12. Especificación caso de prueba manual MT014 . . . . .	81
4.13. Especificación caso de prueba manual MT015 . . . . .	82
4.14. Especificación caso de prueba manual MT016 . . . . .	83
5.1. Resultados de la evaluación de la funcionalidad . . . . .	89
5.2. Escala de rangos de <i>SUS Score</i> [32] . . . . .	90
5.3. Resultados de la evaluación de la utilidad . . . . .	91
5.4. Resultados de la evaluación de la utilidad del manual de usuario . . .	93

## RESUMEN

# 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 [22]. 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 [13].

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 NSGAI.

## 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 basado en la arquitectura del framework multiobjetivo Jmetal [9].
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 buscar soluciones a dos de los problemas existentes en las redes de agua potable.

Los problemas que se abordaran 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 [12]. El problema de diseño a tratar consiste en la optimización de los costo de inversión variando el diámetro de las tuberías (*Pipe Optimizing*). Este problema sera abordado utilizando el Algoritmo Genético (GA).
- **Problema de operación:** Los problemas de construcción buscan optimizar las configuraciones de una red ya construida [12]. El problema de operación a tratar consiste en la optimización de costos energéticos y el número de encendidos y apagados de las bombas (*Pumping Schedule*). Este problema se aborda desde el

enfoque multiobjetivo utilizando el algoritmo *Non-Dominated Sorting Genetic Algorithm*, versión II (NSGAI).

Se seleccionaron estos problemas 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.

Tanto GA como NSGAI, permiten la utilización de distintos operadores de cruce y mutación que también serán implementados para ser utilizados.

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 serán los siguientes:

- Esta herramienta sólo podrá 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 permitirá la carga y la visualización de la red gráficamente.
- El sistema permitirá 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 únicamente contará con dos algoritmos implementados por defecto los cuales serán el Algoritmo Genético (GA) y *Non-Dominated Sorting Genetic Algorithm II* (NSGAI).

- El sistema sólo resolverá dos tipos de problemas de optimización, uno monoobjetivo y el otro multiobjetivo. El Algoritmo Genético se utilizara para generar soluciones para el problema monoobjetivo con el objetivo de optimizar el costo de inversión de las tuberías. Por otro lado, el algoritmo NSGAI aborara el problema multiobjetivo con el fin de optimizar los costos energéticos y el régimen de bombeo.
- El sistema permitirá visualizar él o los resultados obtenidos al finalizar la ejecución del algoritmo.
- El sistema permitirá utilizar una solución obtenida, a través de los algoritmos metaheurísticos, para generar un nuevo archivos de configuración de red.
- El sistema permitirá generar archivos con las soluciones obtenidas para cada problema, es decir, el valor de los objetivos y las variables de decisión involucradas.
- El sistema permitirá graficar visualmente las soluciones en un plano cartesiano.
- La gráfica únicamente estará 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 si y que se realizan en cierto orden definiendo un flujo de trabajo que permite llevar a cabo el desarrollo de un proyecto [24].

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 [1].

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 [34, 20, 18, 1].

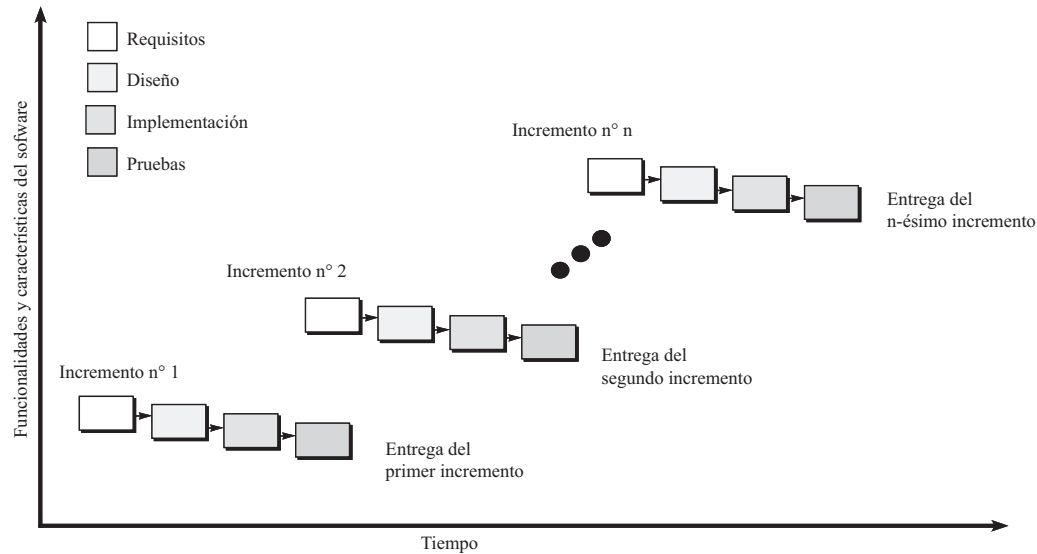


Figura 2.1: Metodología iterativa e incremental [24]

## 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 [29] son una de las metodologías que puede ser usada 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 como se realizaran el análisis de estos.

- (iv) **Definición de características a evaluar:** Consiste en establecer qué es lo que estamos interesados en evaluar del elemento sobre el que se aplica el estudio de caso.
  - (v) **Definición de sujetos de prueba:** Consiste en indicar cual sera 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 sujeto 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 la implementación del software

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

#### 2.3.1. Java

Java es un lenguaje de programación de alto nivel orientado a objetos y de propósito general. 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 [11]. Además, este lenguaje también incorpora una característica conocida como el recolector de basura, que se encarga de limpiar la memoria de

objetos que ya no están siendo utilizados. Fue anunciado por Sun Microsystems en Mayo de 1995 [30].

La elección de Java como lenguaje de programación para desarrollar este proyecto se debe a la familiaridad que el grupo de trabajo que ha solicitado este sistema tiene con el lenguaje.

### 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 [6].

### Java Annotation

Característica de Java para agregar metadatos a elementos del lenguaje como las clases, métodos, parámetros, etc [25]. 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 [7] es la técnica que permite modificar el comportamiento de un elemento del lenguaje, 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. Inversión de control

La inversión de control [31] 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 [15].

### 2.3.3. Inyección de dependencias

La inyección de dependencias (DI) [31], 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 [15] la inyección de dependencias presenta los siguientes problemas [15]:

- 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.

### 2.3.4. Epanet Programming Toolkit

Librería de enlace dinámico que permite realizar simulaciones computacionales 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 [27]. 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 [28].

## 2.4. Redes de agua potable

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:

**Presión:** Fuerza ejercida sobre una superficie. En este caso se refiere a la fuerza que el agua ejerce durante su recorrido. Se expresa en el sistema internacional a través de metros columna de agua (mca)[Insertar cita].

**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 ( $m^3/s$ ).

**Factor de fricción:** Coeficiente adimensional que especifica la rugosidad de la tubería [23].

**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.

#### 2.4.1. Red de distribución de agua

Conjunto de elementos enlazados de tal manera que permite suministrar cierta cantidad de agua a una presión establecida [21]. En la Figura 2.2 se puede apreciar los componentes que conforman la red de agua potable.

##### Componentes

**Nodos de consumo:** Son los puntos o extremos de una tubería, los cuales también permiten que estas se unan. Estos nudos pueden actuar como nudos de demanda a través de los cuales el flujo abandona la red.

**Reservorio:** Es una fuente de alimentación externa. (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.)

**Deposito:** Son elementos con la capacidad de almacenar agua.

**Tuberías:** Es el medio por el cual transita el agua desde un lugar a otro.

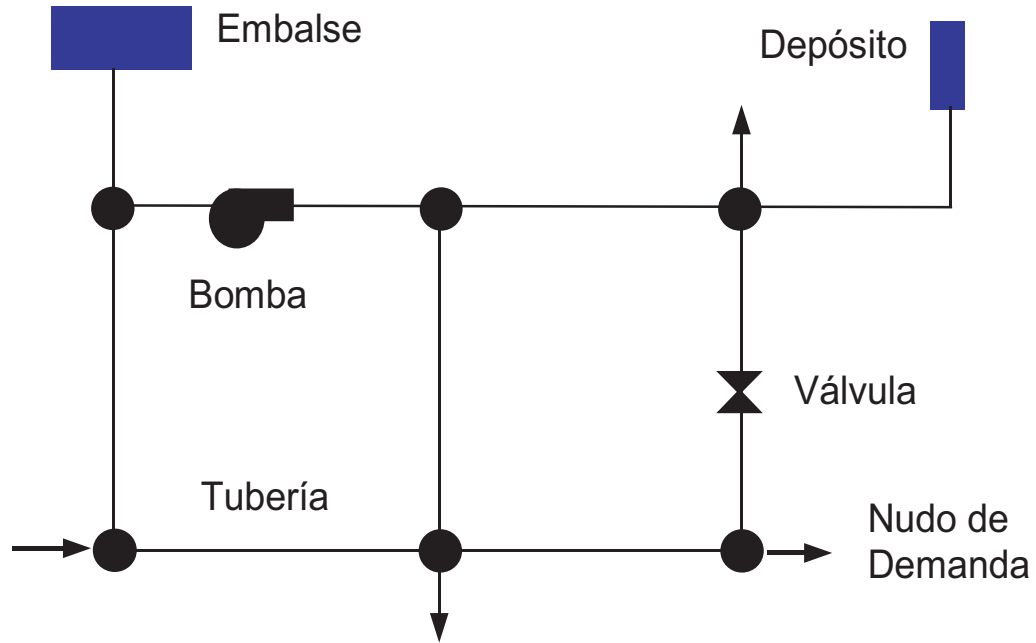


Figura 2.2: Componentes físicos de un sistema de distribución de agua [27]

**Bombas:** Son el componente que permiten impulsar el líquido con el fin de elevarlo a una posición superior.

**Válvulas:** Elementos que limitan la presión o el caudal que transita en un punto de la red.

## 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

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

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



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 [35]. 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** [35]. 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 2.7.3. En este punto se debe tener en cuenta que los objetivos planteados deben encontrarse en contradicción.

Debido a la definición de las restricciones es posible dividir el espacio de búsqueda en dos regiones [5]:

- 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.

## 2.6. 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 está garantizado [35, 26].

## 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 esta. La diferencia entra

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 [35, 4, 16].

### 2.7.1. 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 [4]. En la Figura 2.3 muestra un pseudocódigo de los pasos generales de un algoritmo evolutivo.

---

#### Algorithm 1: Algoritmo Evolutivo

---

```

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

```

---

Figura 2.3: Pseudocódigo Algoritmo Evolutivo

Primero, se crea la población inicial. Luego, 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 número 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 reemplaza la población inicial con la descendiente.

### Población

Conjunto de soluciones candidatas sobre las cual 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. Una solución en la población se conoce como individuo [14].

Los individuos pueden ser representados de diversas maneras, entre ellas se encuentra la representación binaria (1 y 0), la real (Los 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,

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

### Selección

La selección es un mecanismo utilizado por los algoritmos evolutivos para escoger a los individuos mas aptos los cuales serán usados para la reproducción [14]. Existen numerosos algoritmos de selección que pueden ser utilizados en los algoritmos evolutivos.

### Cruzamiento

El cruzamiento es un mecanismo usado para generar nuevas soluciones a partir de dos o mas individuos seleccionados [3, 4].

### Mutación

La mutación es un operador el cual permite mantener la diversidad en la descendencia [4] realizando modificaciones en ciertas partes de la solución.

#### 2.7.2. Algoritmo Genético

El Algoritmo Genético es una estrategia de búsqueda de soluciones. 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 [14]. Este algoritmo pertenece a la categoría de algoritmos evolutivos y por lo tanto puede usar el mismo esquema presentado en la Figura 2.3.

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

### 2.7.3. Conceptos para la 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 [33]:

#### Dominancia de Pareto

Sean  $u$  y  $v$  vectores pertenecientes a  $\mathbb{R}^n$ , 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.

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

#### Optimo de Pareto

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

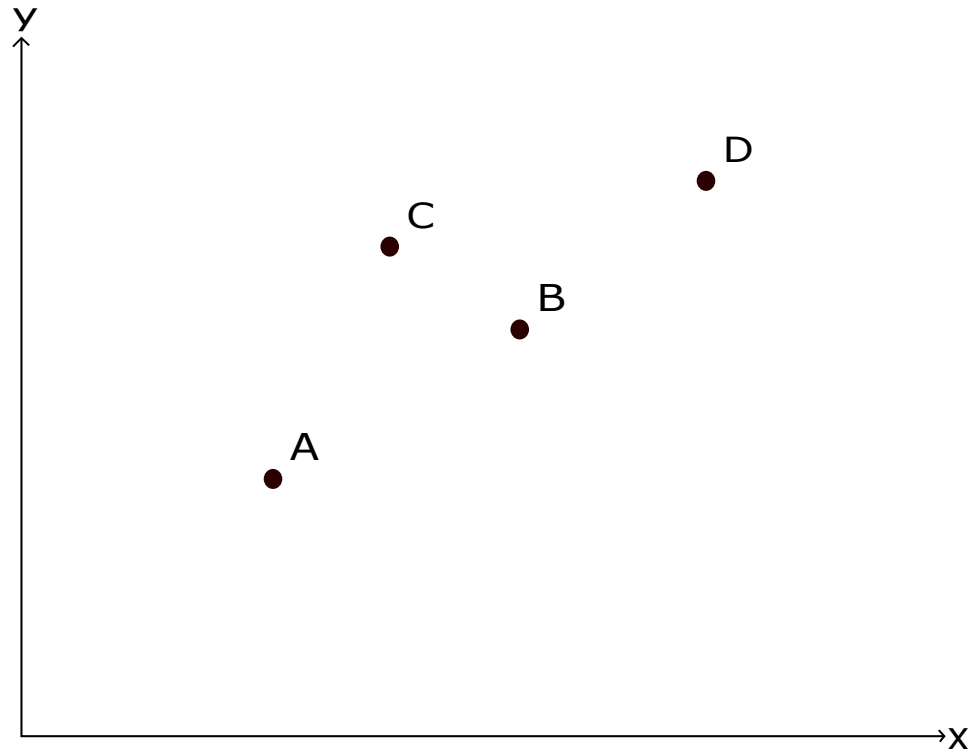


Figura 2.4: Ejemplo de dominancia y Óptimo de Pareto

### 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.

#### 2.7.4. Algoritmo NSGAI (Non-Dominated Sorting Genetic Algorithm II)

El algoritmo NSGAI [8] 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, NSGAI añade conceptos y operadores adicionales los cuales permiten mejorar su rendimiento y la calidad de las soluciones obtenidas. NSGAI puede ser implementado siguiendo los mismos pasos de el algoritmo evolutivo mostrados en la Figura 2.3, utilizando la función de remplazo mostrada en la Figura 2.6.

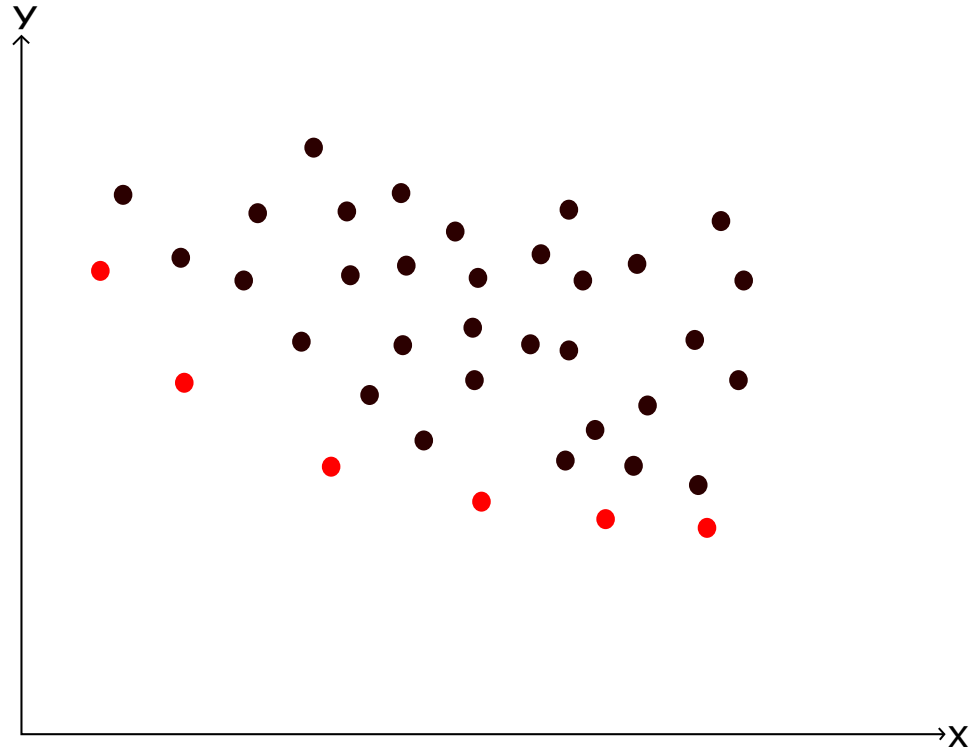


Figura 2.5: Ejemplo frente de Pareto

En la Figura 2.6 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 2.7.

A continuación se procederá a explicar los operadores adicionales presentados en [8] y que son utilizados por la función de remplazo del algoritmo NSGAII en la

---

**Algorithm 2:** Función de remplazo para el algoritmo NSGAII

---

```

1 Function remplazar(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  ordenar ( $F_i, \prec_n$ )
17  /* Elegir los primeros  $N - |nuevaPoblacion|$  */
18  nuevaPoblacion  $\leftarrow$  nuevaPoblacion  $\cup F_i[1 : N - |nuevaPoblacion|]$ 
19  retornar nuevaPoblacion
20 fin

```

---

Figura 2.6: Pseudocódigo de la función de remplazo utilizada en el algoritmo NSGAII [8]

Figura 2.6.

### Ordenamiento de soluciones en frentes no dominados

Uno de los procedimientos presentados en la Figura 2.6 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 2.8 se muestra un ejemplo con tres frentes.

El algoritmo para llevar a cabo esto se presenta en la Figura 2.9 y consiste en lo siguiente:

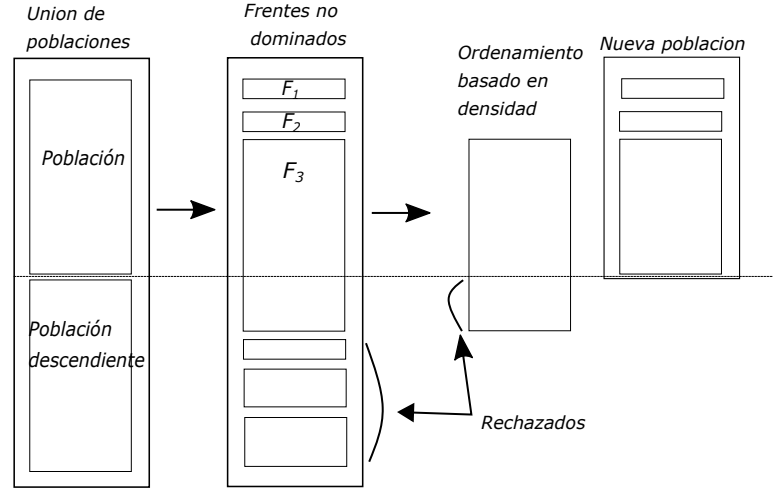


Figura 2.7: Procedimiento NSGAII [8]

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.

### Densidad de estimación (Crowding Distance)

Otra función presente en la Figura 2.6 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 2.10 la densidad de



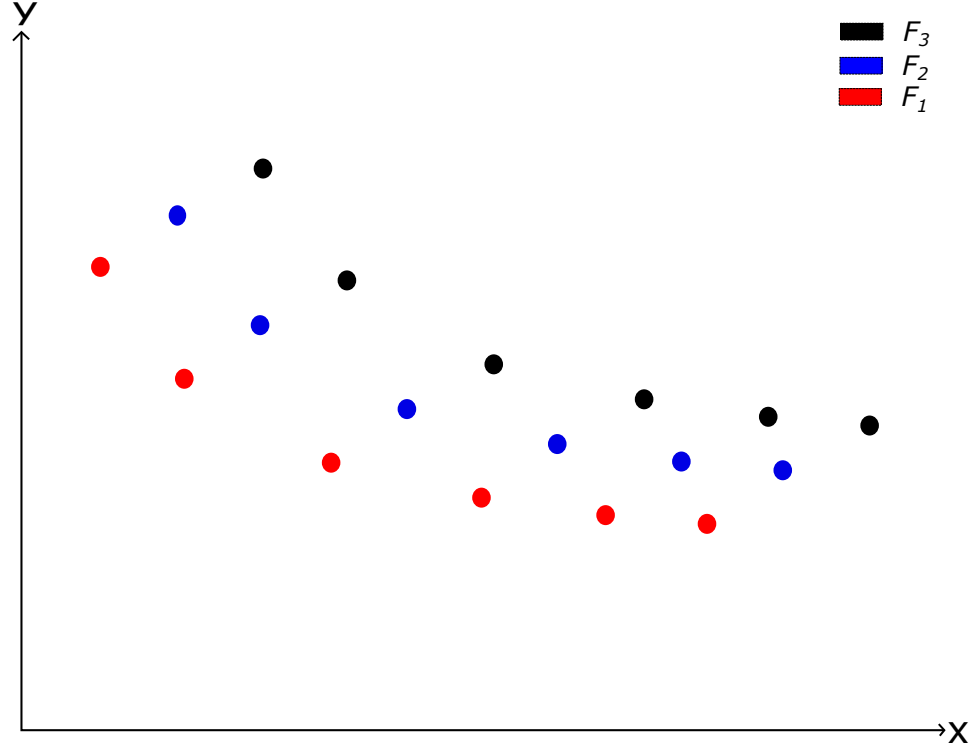


Figura 2.8: Frentes no dominados

la solución  $i$  corresponde a la suma de la longitud del lado mayor y del lado menor del rectángulo.

El procedimiento para esta función se muestra en la Figura 2.10 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 ultima solución.

### Comparador de densidad (Crowding 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 [8], se define el orden dado por el operador de densidad ( $\prec_n$ ) como:  $i \prec_n j$ , si  $(i_{rango} <$

$j_{rango}$ ) o  $((i_{rango} = j_{rango}) \text{ o } (i_{distancia} > j_{distancia}))$ .

## 2.8. 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 [10] 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 NSGAI. 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 costo 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 [19] 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 [21] 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 está 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.

**Algorithm 3:** Función de ordenación en frentes no dominados

---

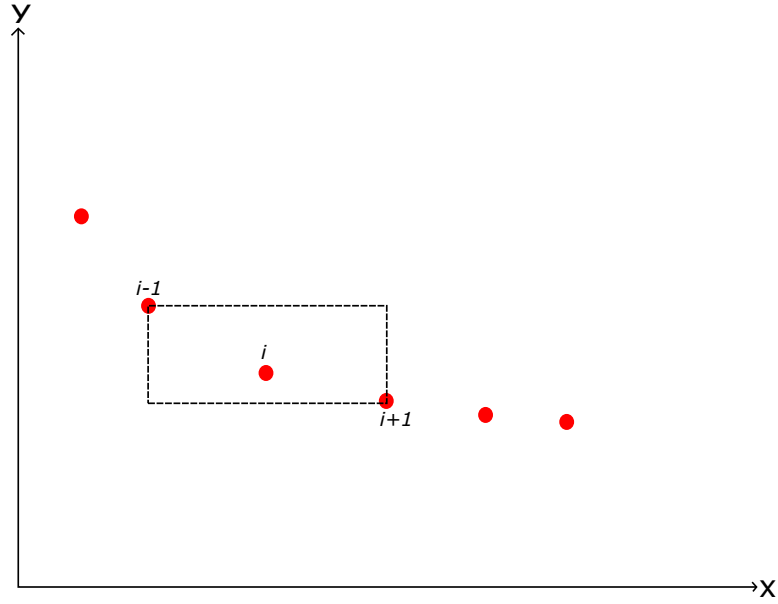
```

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

```

---

Figura 2.9: Pseudocódigo de la función de ordenamiento utilizada en NSGAII [8]

Figura 2.10: Cálculo de la densidad de estimación al rededor de la solución  $i$  [8]**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      $\mathcal{I}[i]_{\text{distancia}} \leftarrow 0$ 
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 ultima solución el valor  $\infty$  */
11     $\mathcal{I}[1]_{\text{distancia}} \leftarrow \mathcal{I}[l]_{\text{distancia}} \leftarrow \infty$ 
12    for  $i \leftarrow 2$  to  $l - 1$  do
13      /* Asigna la distancia a las soluciones restantes */
14       $\mathcal{I}[i]_{\text{distancia}} \leftarrow \mathcal{I}[i]_{\text{distancia}} + (\mathcal{I}[i + 1].m - \mathcal{I}[i - 1].m) / (f_m^{\text{max}} - f_m^{\text{min}})$ 
15    end
16  end
17 fin

```

---

Figura 2.11: Pseudocódigo de la función de asignación de densidad [8]

## 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 consiste, de manera general, en disminuir la cantidad de documentación generada por cada fase, flexibilidad para permitir llevar a cabo más de una fase dentro de cada iteración al mismo tiempo y que los roles de analista, diseñador, implementador y tester sean realizados por una sola persona.

Las tareas a desempeñar por cada fase consisten en:

### **Análisis:**

- Captura de requisitos: Durante esta etapa se reúne con el cliente ya sea a físicamente o de manera remota y 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 en 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, entre otras maneras.
- 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: Consiste en codificar con el fin de implementar 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 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 ciertas 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 a utilizar, 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. Específicamente, estos contienen 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 vs 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 [2].
- 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 la 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 que elemento se llevan a cabo.

La razón por la que se utiliza esta metodología sobre otras es porque el producto resultante de este proyecto esta pensado para servir como base para futuros trabajos. Debido a esto es necesario documentar correctamente los detalles de 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 Civil, Daniel Mora Melia. Él, junto a un grupo de expertos de diversas áreas, presentaron y publicaron un artículo del proyecto JHawanet [13]. Dicho artículo, presenta la integración de dos librerías independientes, JMetal y Epanet, como herramienta para llevar a cabo optimizaciones sobre RDA. JMetal [9] es un Framework de Java, orientado principalmente a la optimización multiobjetivo y es usado como motor de optimización. Mientras que Epanet [28] es una herramienta la cual permite realizar simulaciones de 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.

### 4.2. Casos de prueba

Los casos de prueba son las funcionalidades que se usarán para presentar y demostrar la aplicación de la metodología durante el desarrollo del proyecto.

Se escogerán 3 funcionalidades de prueba las cuales son:

**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 buscar soluciones al problema de la optimización de los costos de construcción de las tuberías usando el Algoritmo Genético.

Esta funcionalidad consiste en buscar soluciones para el problema de optimización de los costos de construcción de las tuberías.

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

Esta funcionalidad consiste en realizar una simulación hidráulica utilizando el archivo de red cargado y visualizar los valores resultantes de los elementos de la red.

### 4.3. Planificación

Como se menciona 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

N° Iteración	Tareas	Fecha Inicio	Fecha término
1	<ul style="list-style-type: none"> <li>- Especificación de requisitos</li> <li>- Escoger arquitectura lógica y física</li> </ul>	26/08/2019	14/10/2019
2	<ul style="list-style-type: none"> <li>- Implementar problema monoobjetivo (Pipe Optimizing)</li> <li>- Implementar Algoritmo Genético</li> <li>- Implementar operadores de selección y reproducción</li> </ul>	14/10/2019	11/11/2019
3	<ul style="list-style-type: none"> <li>- Crear interfaces de usuario</li> <li>- Guardar soluciones</li> </ul>	11/11/2019	20/01/2020
4	<ul style="list-style-type: none"> <li>- Implementar problema multiobjetivo (Pumping Schedule)</li> <li>- Implementar algoritmo NSGAI</li> </ul>	27/01/2020	24/02/2020
5	<ul style="list-style-type: none"> <li>- Permitir realizar múltiples repeticiones de un algoritmo</li> <li>- Permitir realizar la simulación usando los valores por defecto de la red</li> </ul>	02/03/2020	04/05/2020
6	<ul style="list-style-type: none"> <li>- Agregar símbolos al dibujo de la red</li> <li>- Exportar a excel</li> <li>- Agregar menu de configuración</li> </ul>	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.

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

N° 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 requisitos iniciales de la aplicación fueron capturados del profesor Jimmy Gutierrez. Posteriormente, estos requisitos fueron priorizados y validados 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.

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.

<b>RU004 – Visualizar red en una interfaz gráfica.</b>	
<b>Descripción:</b>	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.
<b>Fuente:</b>	Jimmy Gutiérrez
<b>Prioridad:</b>	Moderada
<b>Estabilidad:</b>	Intransable
<b>Fecha de actualización:</b>	09/09/2019
<b>Estado:</b>	Cumple
<b>Incremento:</b>	3
<b>Tipo:</b>	Funcional

Cuadro 4.4: Especificación del requisito de usuario RU002.

<b>RU002 – Resolver el problema monoobjetivo (<i>Pipe Optimizing</i>) usando el Algoritmo Genético.</b>	
<b>Descripción:</b>	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.
<b>Fuente:</b>	Jimmy Gutiérrez
<b>Prioridad:</b>	Alta
<b>Estabilidad:</b>	Intransable
<b>Fecha de actualización:</b>	09/09/2019
<b>Estado:</b>	Cumple
<b>Incremento:</b>	2
<b>Tipo:</b>	Funcional

Cuadro 4.5: Especificación del requisito de usuario RU020.

<b>RU020 – Permitir realizar simulaciones hidráulicas utilizando los valores por defectos que vienen en el archivo inp y visualizar los resultados.</b>	
<b>Descripción:</b>	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.
<b>Fuente:</b>	Daniel Mora-Meliá
<b>Prioridad:</b>	Alta
<b>Estabilidad:</b>	Intransable
<b>Fecha de actualización:</b>	27/01/2020
<b>Estado:</b>	Cumple
<b>Incremento:</b>	5
<b>Tipo:</b>	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.

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 software que posee esta arquitectura funciona localmente sin necesidad de interactuar con otros equipos. Esta elección se debe a que en

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

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



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.

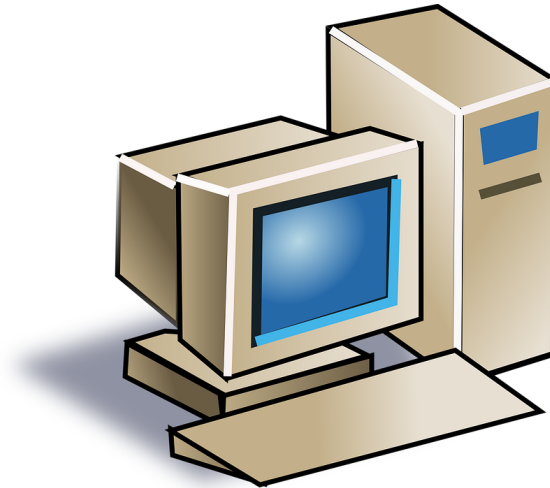


Figura 4.1: Arquitectura física monolítica

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 así escalabilidad y mantenibilidad del software. La Figura 4.2 muestra el diagrama de la arquitectura lógica, así como los principales módulos de la aplicación.

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 reducido (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.

El diagrama de la Figura 4.4 muestra el conjunto de clases y los mensajes de las clases que interactúan con la clase *Network* con el fin de visualizar la red.

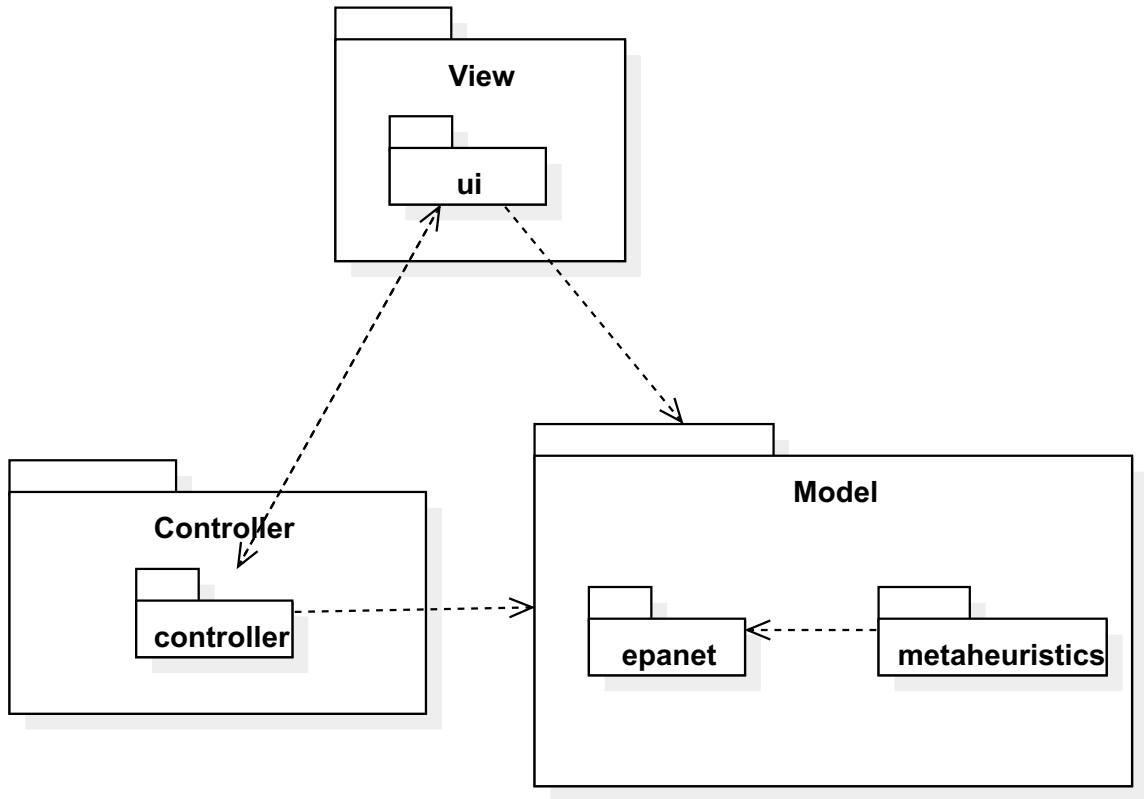


Figura 4.2: Arquitectura lógica Modelo-Vista-Controlador

**Funcionalidad 2:** Esta es una de las funcionalidades ligada al modulo meta-heurística 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 utilizo como base la jerarquía de clases utilizada por el Framework JMetal [9] la cual fue adaptada para ser ocupada por esta aplicación. La Figura 4.5 corresponde a la jerarquía ya mencionada.

Debido a la importancia que tienen las metaheurísticas para la aplicación, se explicará de manera general que realiza cada una de las clases e interfaces presentes en la Figura 4.5.

**Algorithm** es la interfaz desde la que se heredan, ya sea directa o indirectamente, cada uno de los algoritmos metaheurísticos implementados por la aplicación. Su principal método es *runASingleStep* puesto que éste permite realiza una única iteración del algoritmo, permitiendo así que el programa interactué 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 4.6.

***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 usado 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 de un 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.

El diagrama de secuencia de la Figura 4.7 muestra la interacción de las clases con

del modulo metaheurísticas, con el resto de clases de la aplicación.

**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 4.8.

El diagrama de secuencia presentado en la Figura 4.9 muestra la interacción entre las clases necesarias para realizar la simulación hidráulica utilizando los valores del archivo de red cargado.

#### 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 este. Al momento de crear un nuevo algoritmo u operador también estaba presente este problema porque si el usuario quería poner a disposición un nuevo algoritmo para resolver el problema debía realizar cambios en el código de la aplicación.

Finalmente se decidió 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.

El sistema define un flujo de trabajo, el cual se presenta en la Figura 4.10. En él se puede agregar un nuevo problema simplemente definiendo una nueva clase que es usada como plantilla para crear los nuevos experimentos de ese problema con un algoritmo específico. Esta nueva clase debe heredar de una de las subclases de *Registrable*. La jerarquía de herencia de la clase *Registrable* se muestra en la Figura 4.11. En dicha clase se pueden usar anotaciones en el constructor para definir los tipos de las instancias a inyectar así como los valores de configuración que necesita

el experimento. En este caso los tipos corresponden a los operadores a utilizar en el algoritmo del experimento a configurar.

Las anotaciones permitidas en la interfaz *Registrable* que se definieron 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. Esta anotación permite concatenar anotaciones.

**@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 que sera utilizado para inyectar los valores. Dentro de esta anotación se puede usar *@NumberInput* para definir los parámetros que pueden ser inyectados al operador.

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 la el documento de diseño de la aplicación.

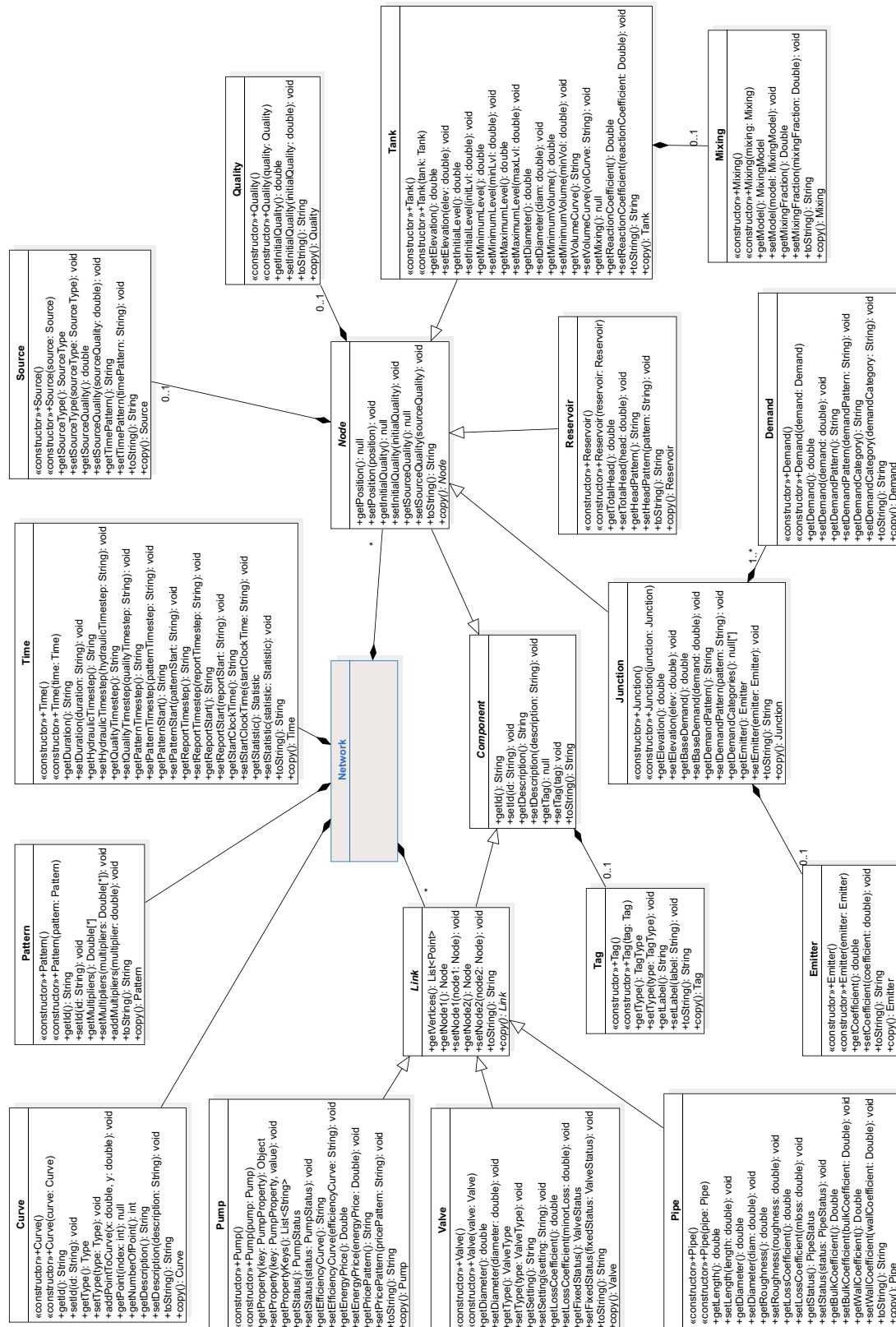


Figura 4.3: Diagrama de clases de la abstracción de la red reducido.

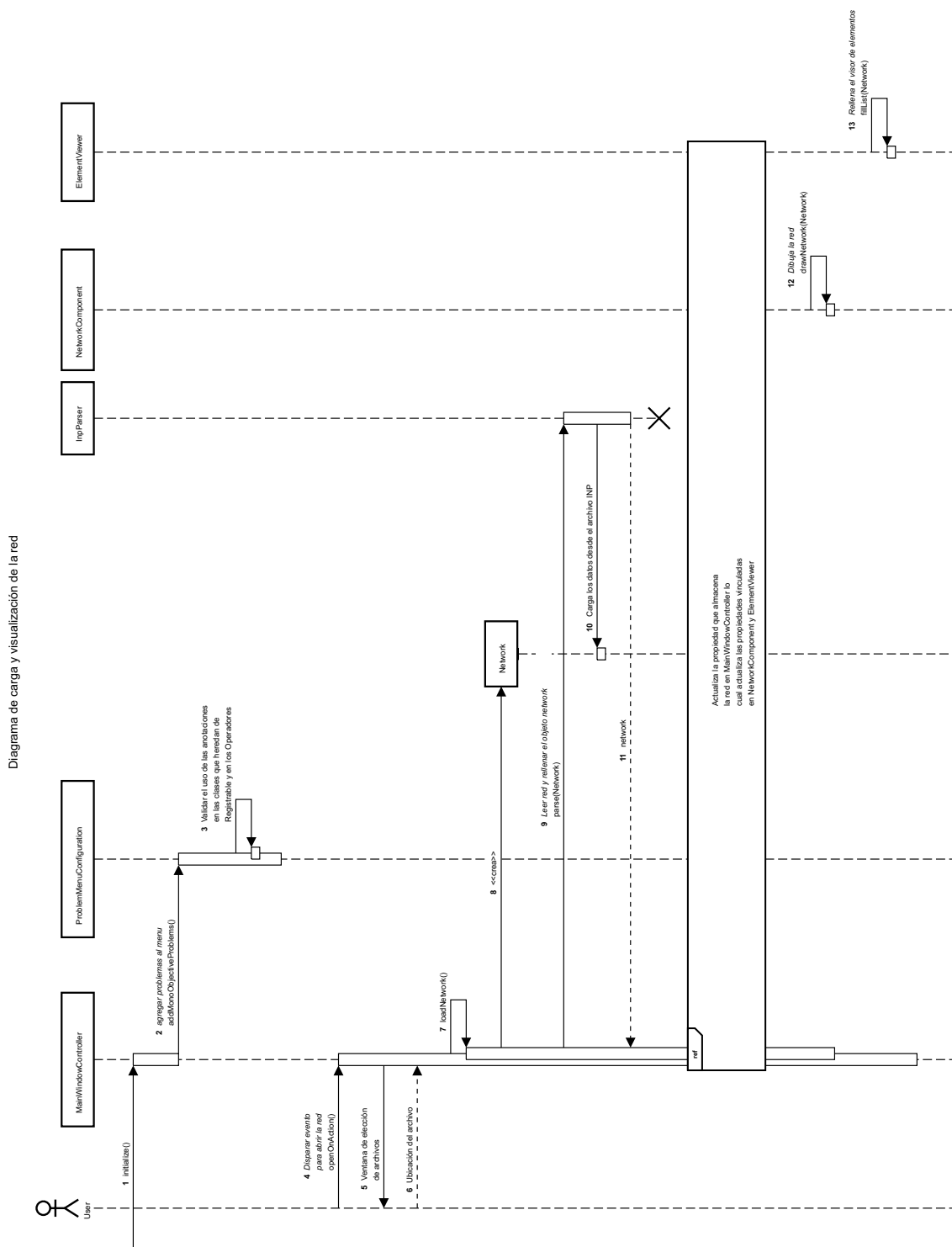


Figura 4.4: Diagrama de secuencia de la carga y visualización de la red



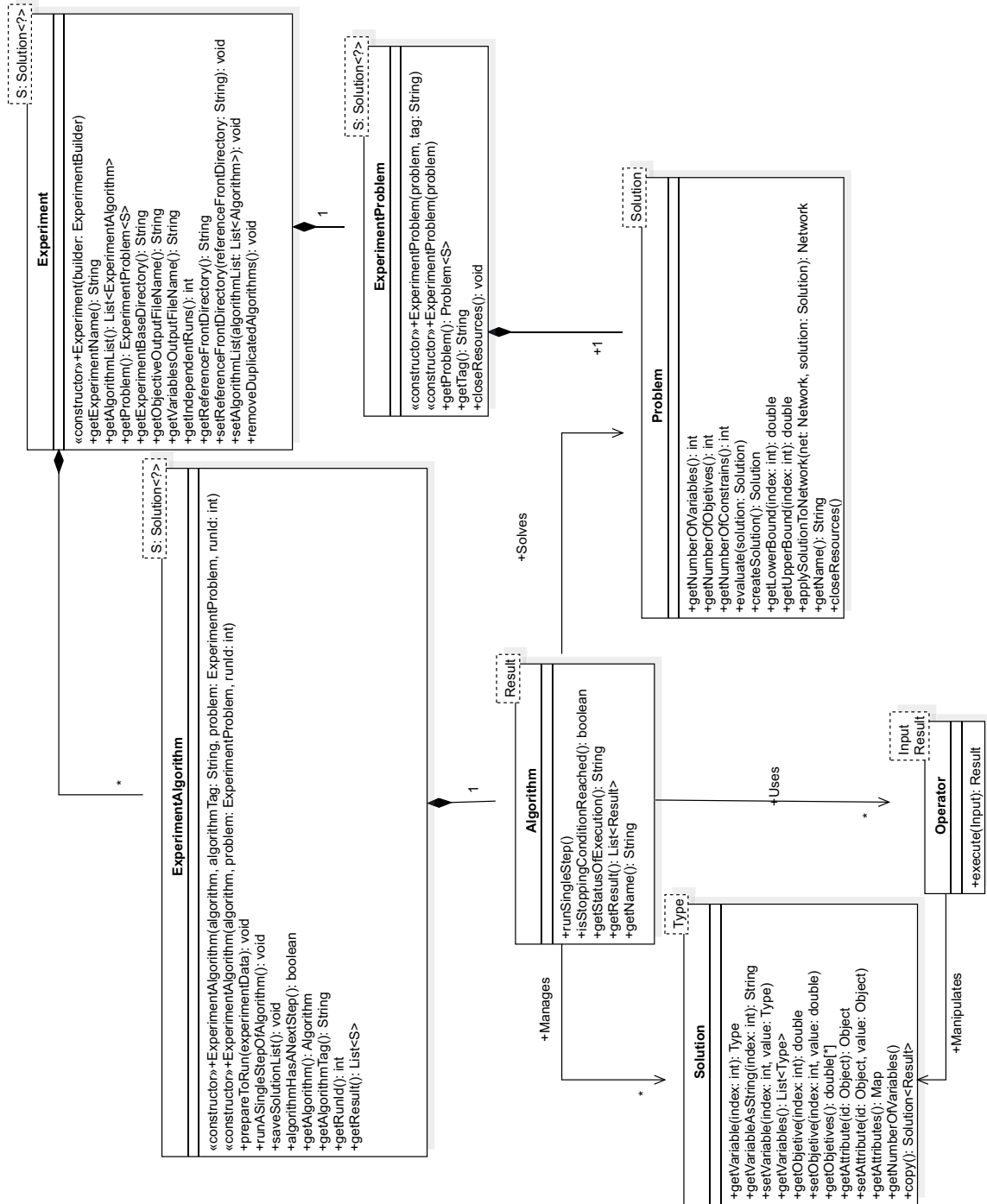


Figura 4.5: Diagrama de clases modulo metaheurística. Modificación a partir del diagrama presentado en [9]

---

```
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 4.6: Código del método *runSingleStep* utilizado con GA y NSGAII

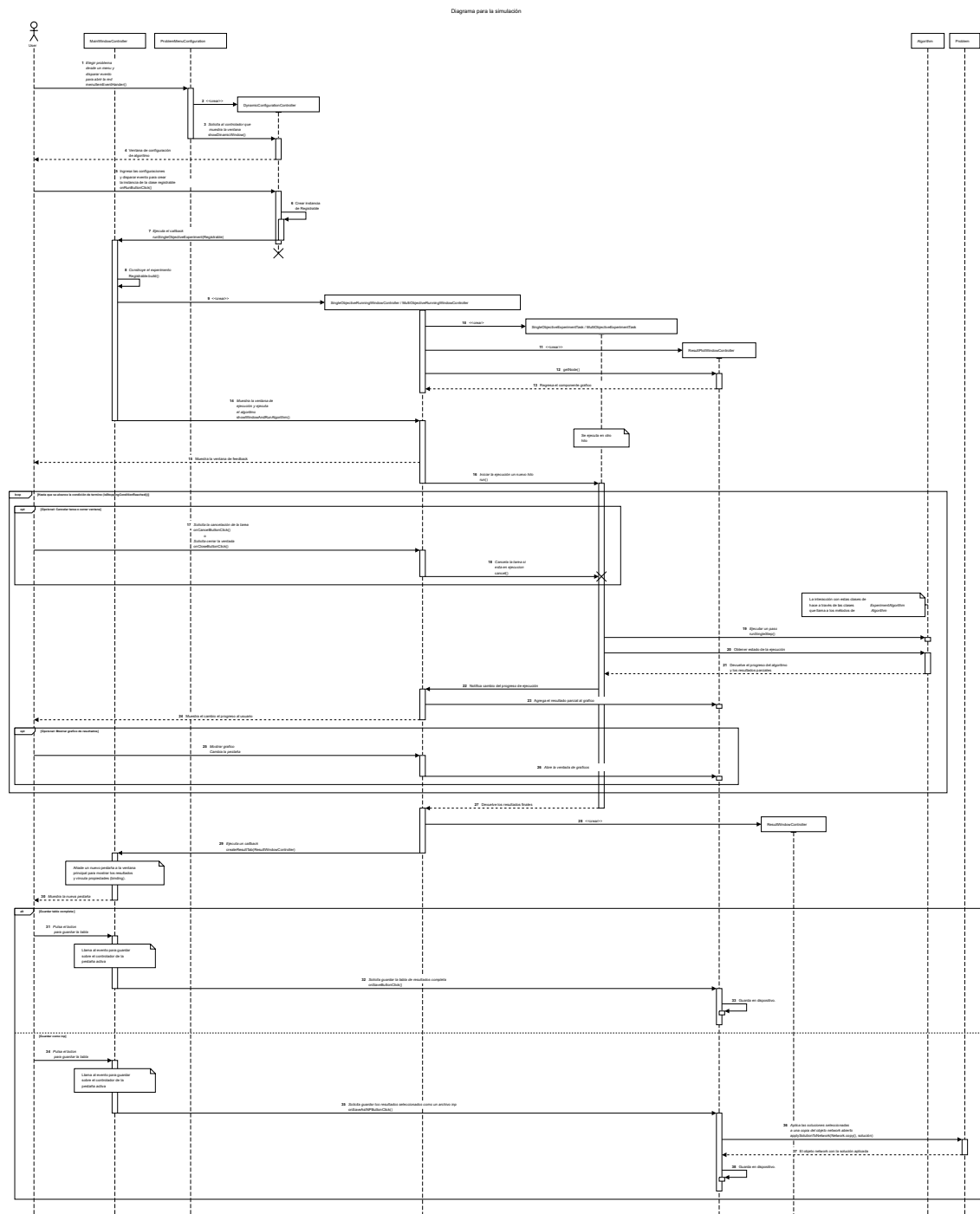


Figura 4.7: Diagrama de secuencia para llevar a cabo la optimizacion de los problemas

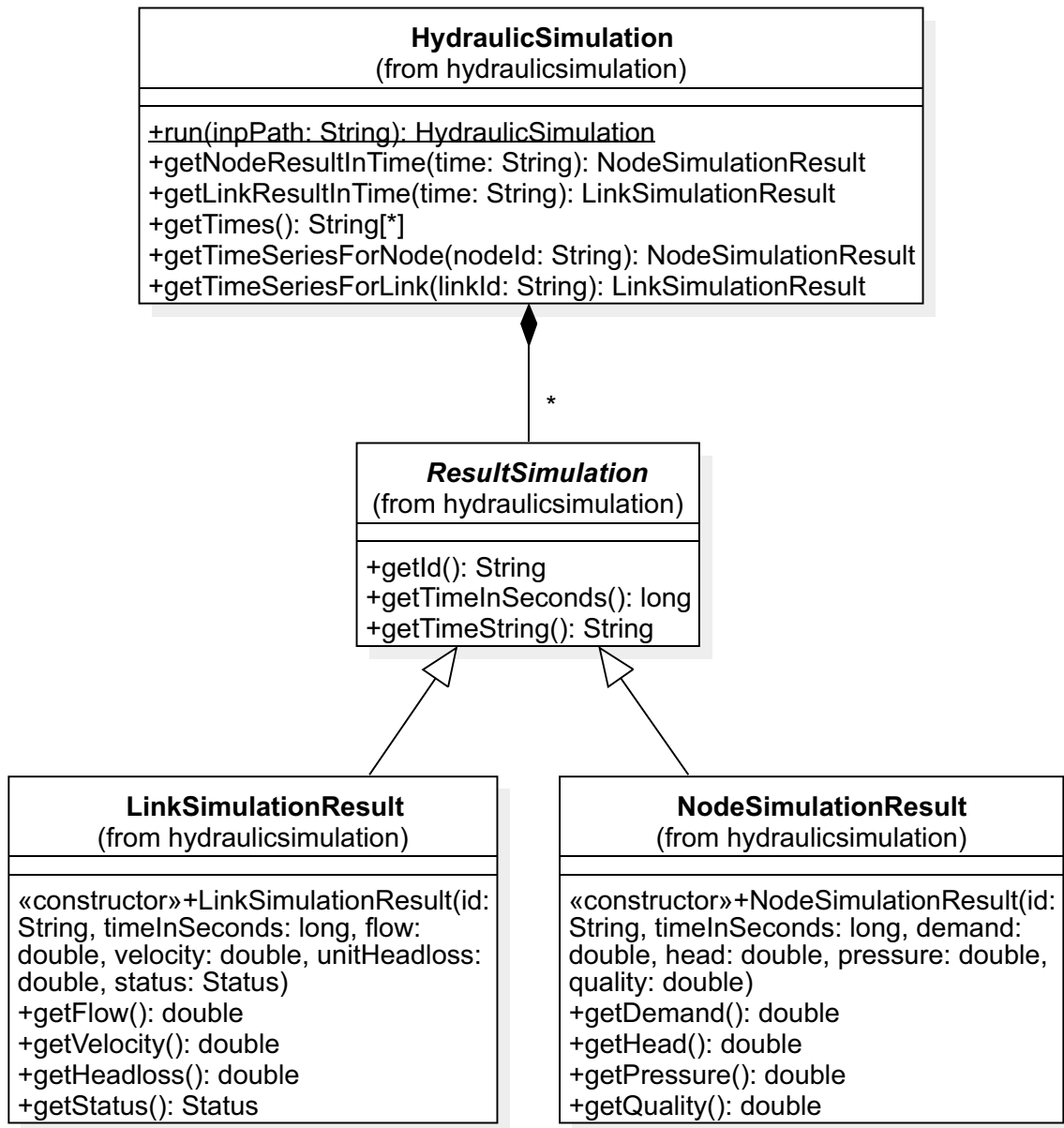


Figura 4.8: Diagrama de clases para la simulación hidráulica

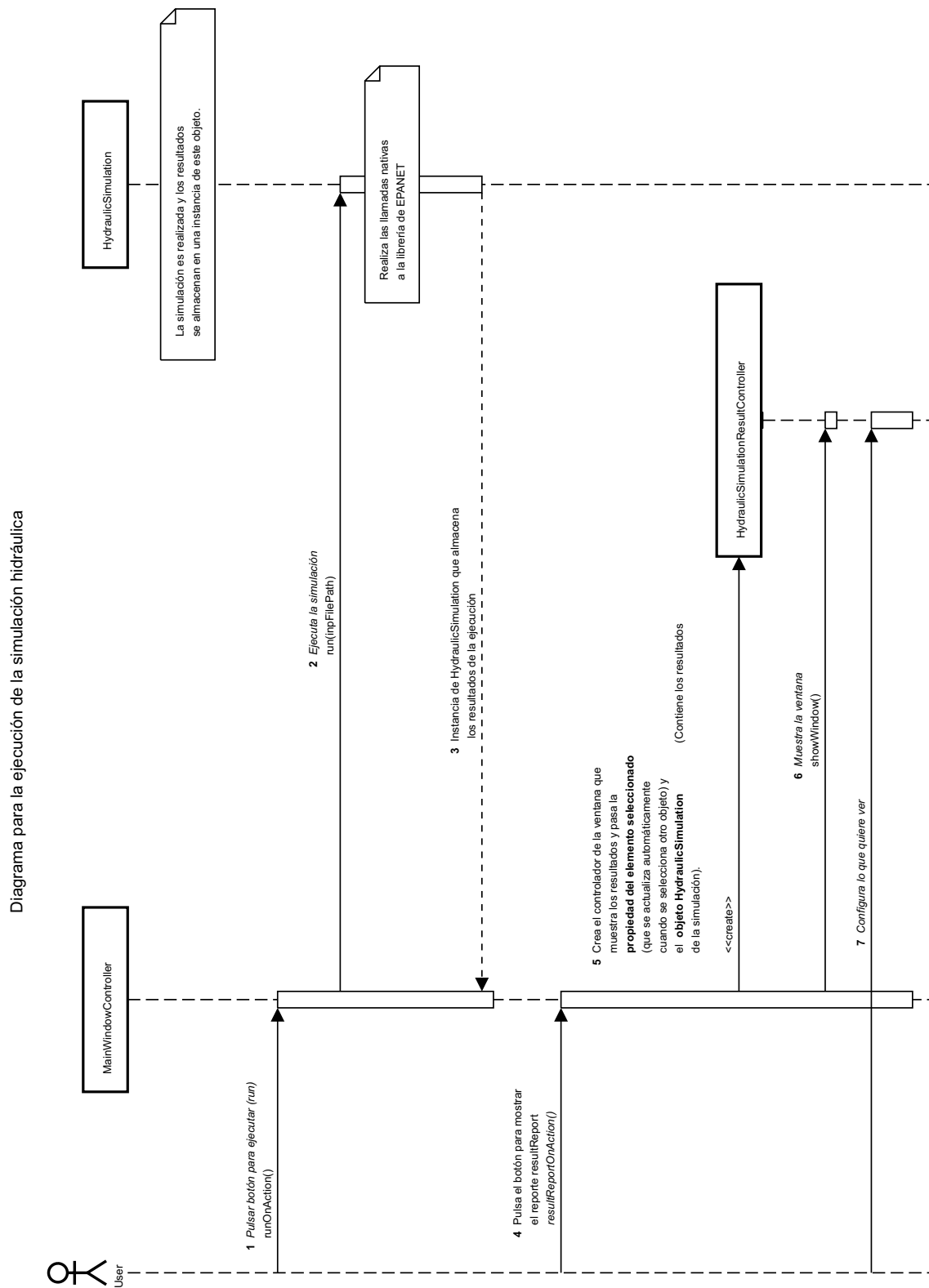


Figura 4.9: Diagrama de secuencia para realizar la simulación hidráulica

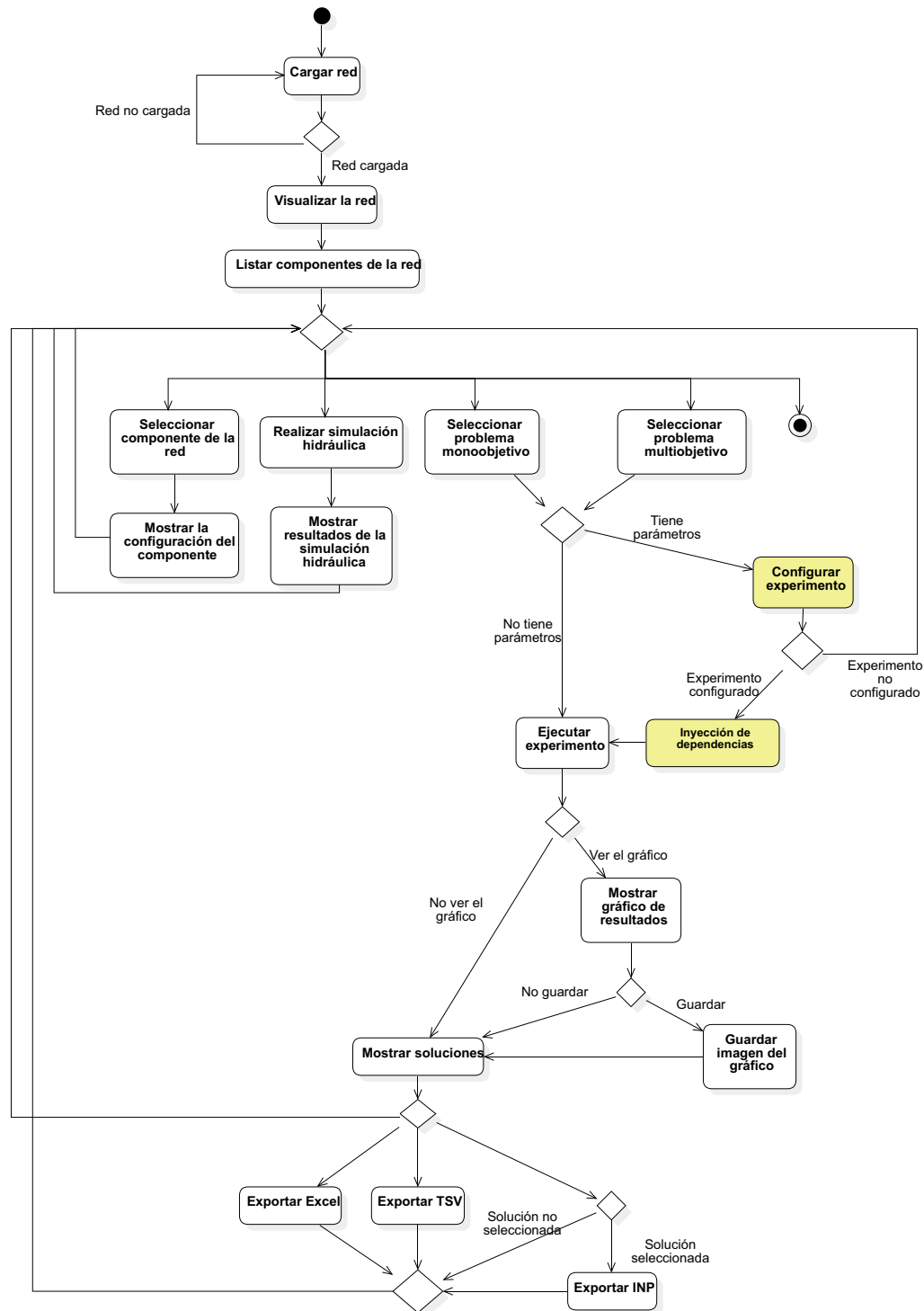
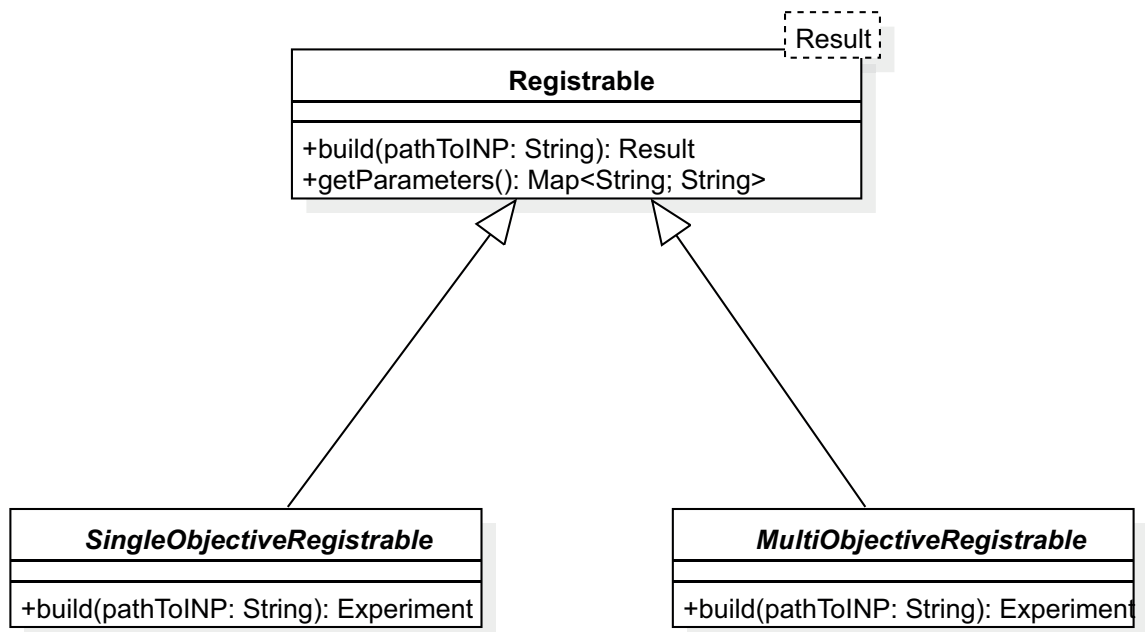


Figura 4.10: Diagrama de actividades de la aplicación. Durante “Configurar Experimento” se leen las anotaciones para construir la interfaz gráfica, una vez configurado el experimento se crean e inyectan las dependencias.

Figura 4.11: Interfaz de clase *Registrable*.

## 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

N° 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 NSGAI.</p> <p>Se implementa la codificación del problema multiobjetivo <b>Pumping Schedule</b>.</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 mono-objetivo (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 4.12 muestra la ventana de visualización de la red.

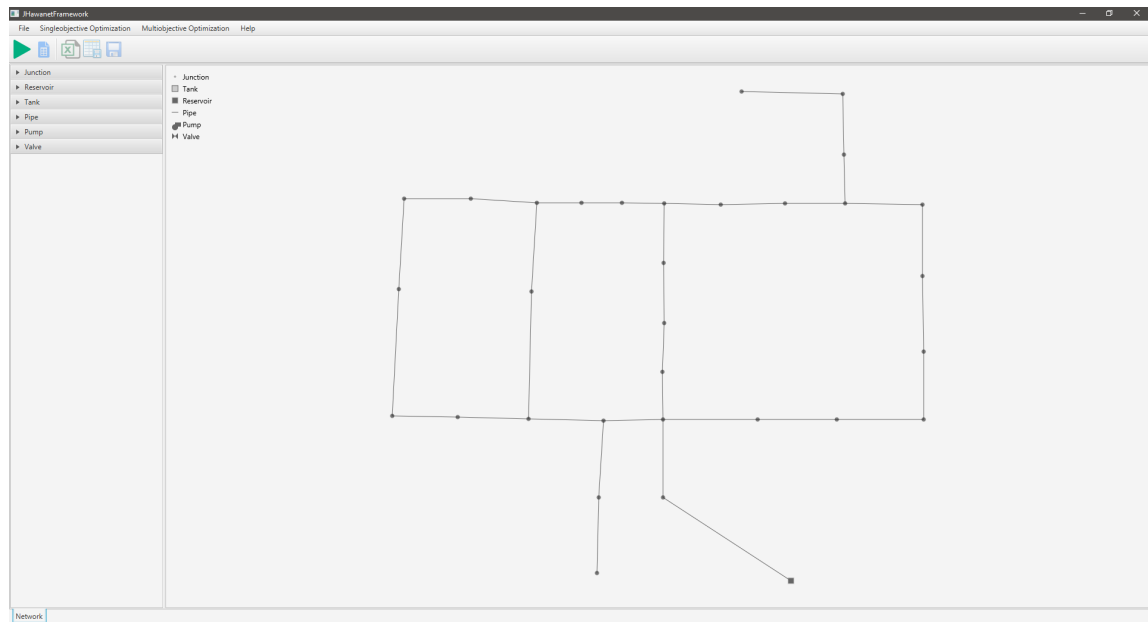


Figura 4.12: Ventana principal de la aplicación donde se visualiza la red

**Funcionalidad 2** : Las Figuras 4.13, 4.14, 4.15, 4.16 y 4.17 muestran las ventana utilizadas durante el cumplimiento de esta funcionalidad.

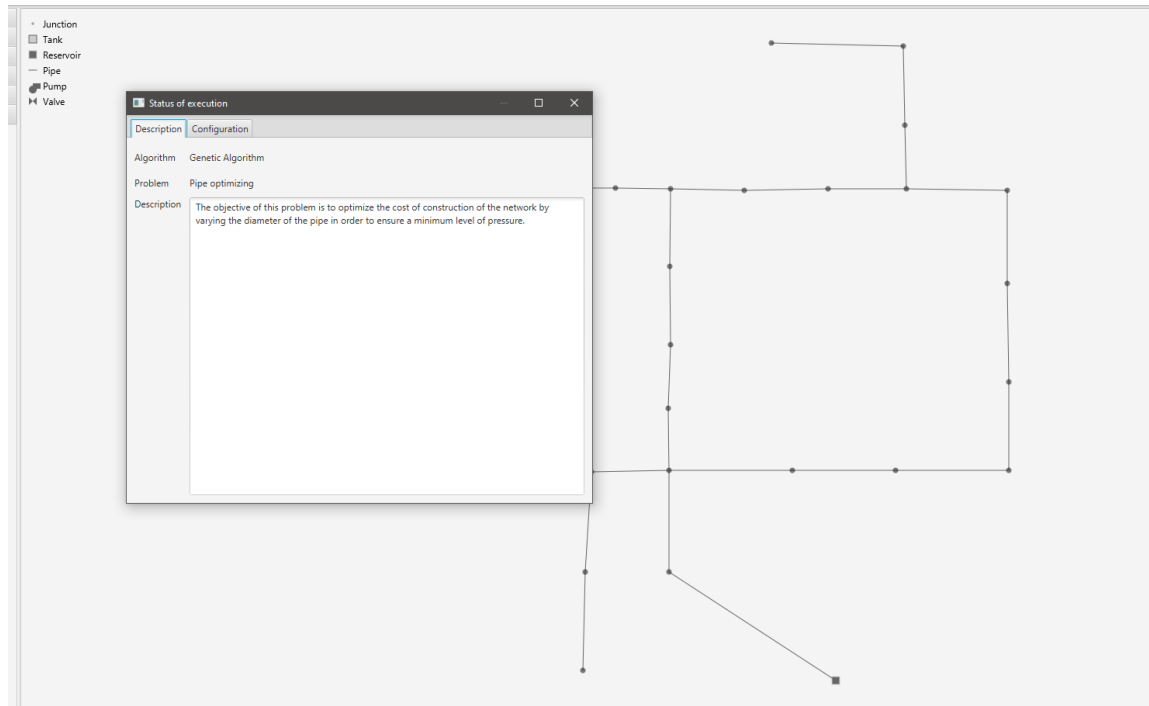


Figura 4.13: Ventana de descripción del problema.

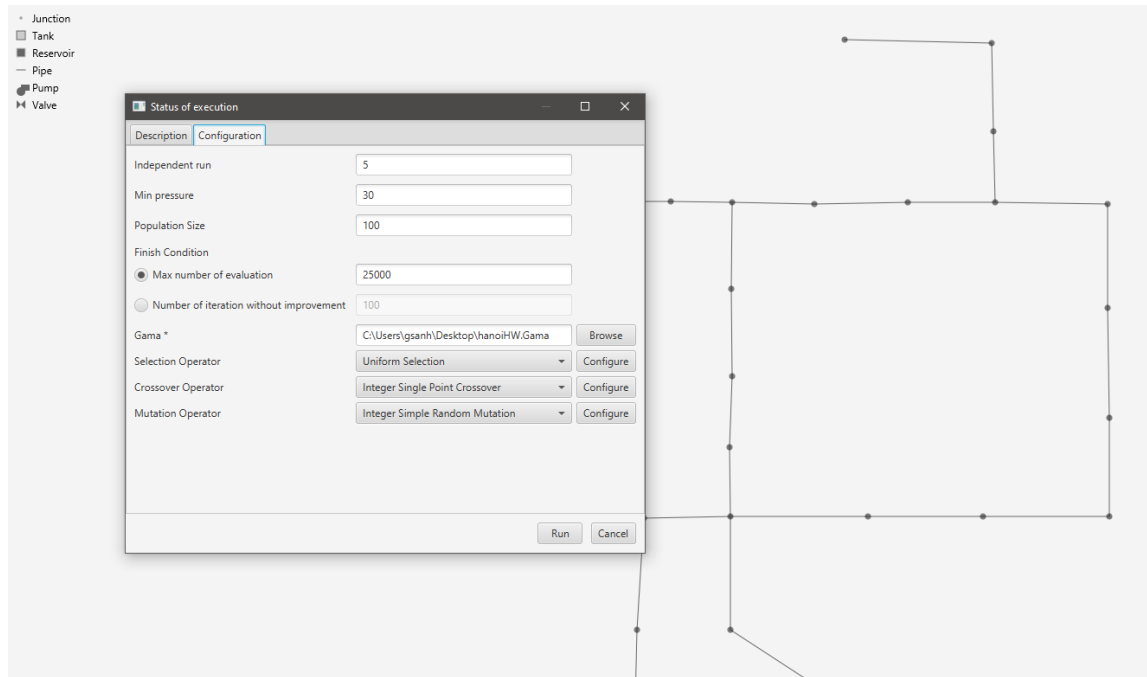
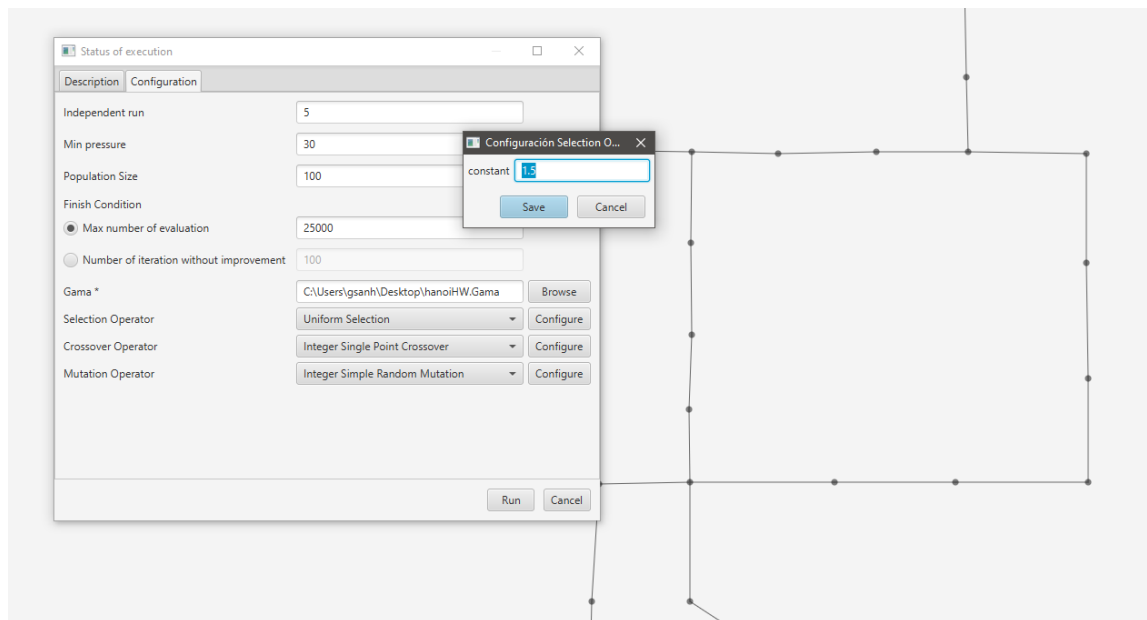


Figura 4.14: Ventana de configuración del problema.

Figura 4.15: Ventana de configuración de parámetros del operador *UniformSelection*

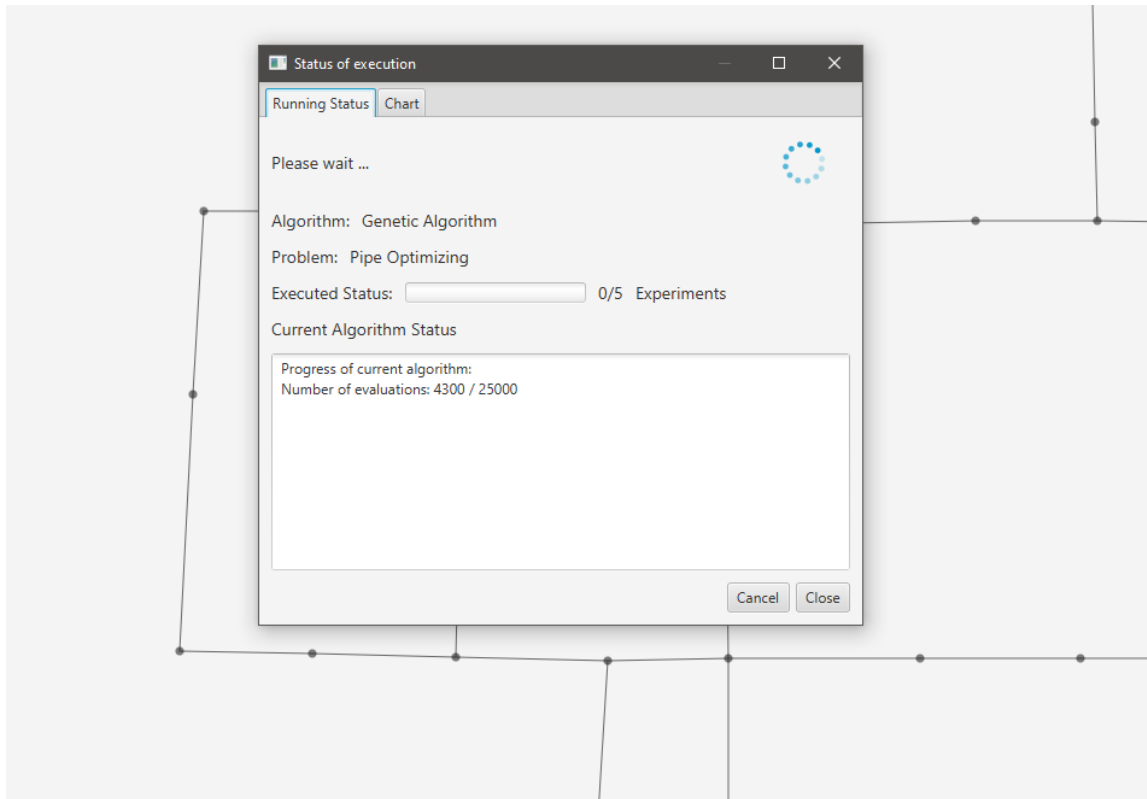


Figura 4.16: Ventana del retroalimentación mostrada durante la ejecución

File

Singleobjective Optimization

Multiobjective Optimization

Help

🏠

📄

🔍

📊

🔗

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 Constran Violation

Number of Generation

Min Pressure

Population

6058409.299171802

6

5

4

5

4

3

5

5

6

6

6

6

6

3

1

6

2

1

1

2

2

4

6

5

3

2

3

1

6

5

6

5

5

4

0.0

250

30

100

7078116.190952501

6

5

4

5

4

4

4

6

6

6

6

6

6

3

2

6

6

3

1

1

1

4

6

4

2

3

2

1

5

5

5

5

4

2

0.0

250

30

100

6788874.769704701

6

5

5

5

4

5

5

5

6

6

6

6

6

3

1

6

5

3

2

1

3

3

6

2

2

1

1

4

5

5

5

5

4

4

0.0

250

30

100

6637209.140444202

6

5

5

4

5

5

4

5

5

6

5

6

6

3

1

6

5

5

2

1

1

2

6

2

1

1

1

3

6

6

5

5

3

5

0.0

250

30

100

8420191.079337701

6

4

5

5

2

2

5

2

5

4

5

6

6

3

2

6

5

5

2

2

2

2

6

2

1

2

1

5

6

6

6

6

5

6

0.0

250

30

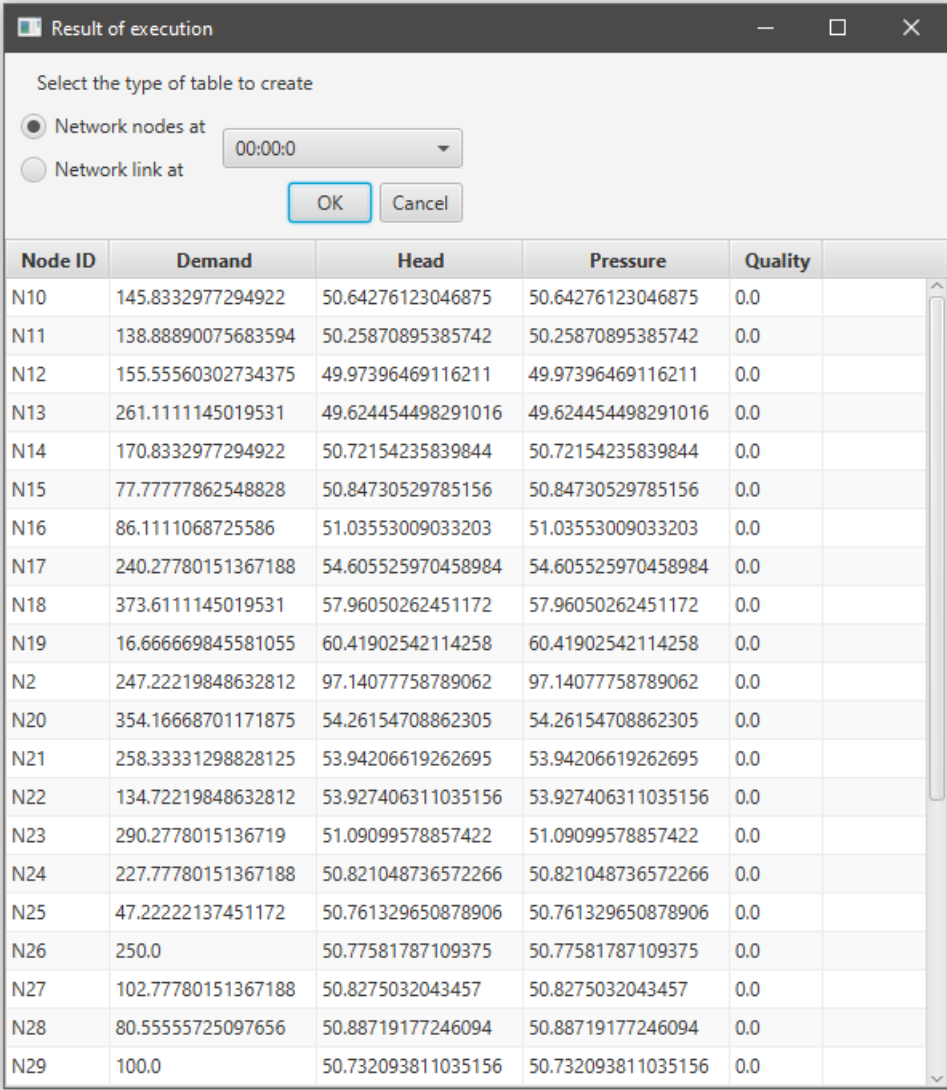
100

Network

0 - Pipe Optimizing-hanoi-Frankenstein origin-2020-06-23

Figura 4.17: Ventana de resultados generada cuando termina la ejecución para el problema monoobjetivo Pipe Optimizing

**Funcionalidad 3** : La Figura 4.18 muestra la ventana con los resultados de una simulación hidráulica para la red cargada.



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 4.18: 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 4.19, genera la interfaz mostrada en la Figura 4.13 y 4.14.

```

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 SEX Crossover", value = IntegerSEXCrossover.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 = 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 4.19: Uso de anotaciones en la clase que hereda *Registrable* para construir la interfaz de configuración del problema



#### 4.6.1. Modelo matemático de los problemas

En esta subsección se presentan los modelos matemáticos de los problemas resueltos, así como la representación de sus soluciones.

##### *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 4.1 presentada en [22] muestra la ecuación a optimizar.

$$\text{Costo de inversión} = \sum_{i=1}^N (C_i \times D_i \times L_i) \quad (4.1)$$

En la ecuación anteriormente presentada el termino  $C_i$  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 menciona anteriormente, el problema debe satisfacer la siguiente restricción:

$$H_i < H_{min} \quad (4.2)$$

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 4.20. 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 inp).

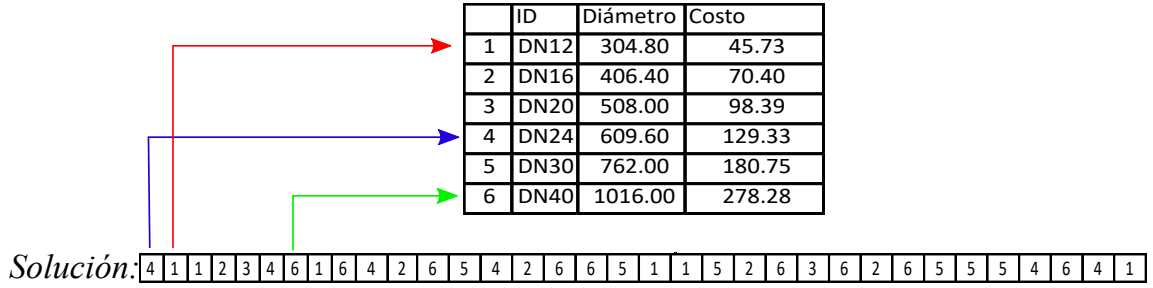


Figura 4.20: Representación de la solución del problema monoobjetivo *Pipe Optimizing*

### ***Pumping Schedule***

*Pumping Schedule* [17, 13] es un problema de operación que tiene como objetivos optimizar tanto el costo energético, así como el costo de operación. 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:

$$C_E(S) = \sum_{n=1}^{NP} \sum_{t=0}^{NT-1} (P_c(t) \times E_c(n, t) \times S(n, t)) \quad (4.3)$$

donde:

- $C_E(S)$ : Costos energético.
- $NP$ : 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:

$$E_c(n, t) = \frac{10^{-3} \times \gamma \times Q(n, t) \times h(n, t)}{e(n, t)} \quad (4.4)$$

donde:

- $\gamma$ : 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 bombeo 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:

$$C_N(S) = \sum_{n=1}^{NP} \sum_{t=0}^{NT-1} r_t \quad (4.5)$$

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 4.4 y 4.5 deben cumplir las siguientes restricciones:

Conservación de la masa:

$$\sum q_{in} - q_{out} = C_j \quad (4.6)$$

donde:

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

Conservación de la energía:

$$\sum h_f - \sum E_p = 0 \quad (4.7)$$

donde:

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

Pérdida de carga por fricción:

$$h_f = \frac{10,67 \times L_q^{1,85}}{CH^{1,85} \times D^{4,87}} \quad (4.8)$$

donde:

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

Presión mínima:

$$H_i < H_{min} \quad (4.9)$$

donde:

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

Caudal:

$$Q_{i,t} \geq Q_i^{max} \quad (4.10)$$

donde:

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

Nivel de depósito:

$$TS_{i,NT} \geq TS_{i,0} \quad (4.11)$$

donde:

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

En la Figura 4.21 muestra como se codifica la solución a este problema. 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 4.5, así como para obtener el estado de la bomba en el periodo  $t$  de la ecuación 4.4 referente al termino  $S(n, t)$ .

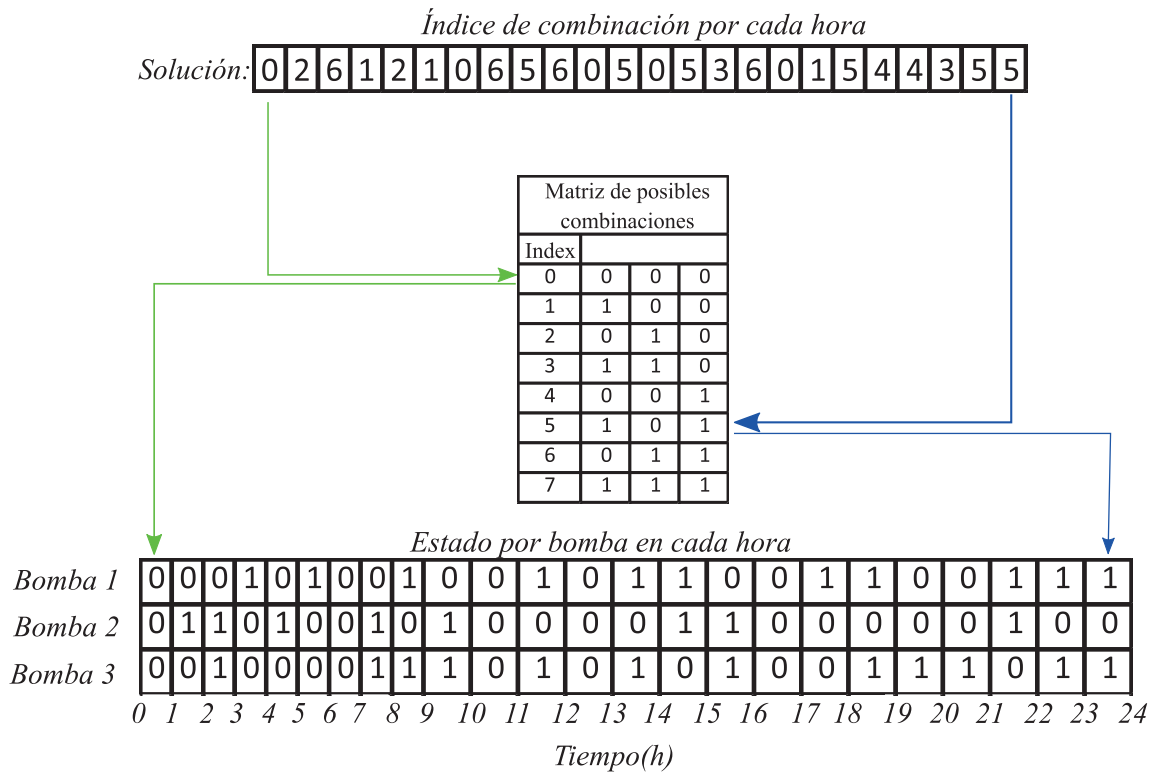


Figura 4.21: Representación de la solución problema multiobjetivo *Pumping Schedule*

## 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.

Cuadro 4.8: Especificación caso de prueba manual MT001

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

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

Cuadro 4.9: Especificación caso de prueba manual MT002

<b>Test ID:</b>	MT002
<b>Título:</b>	Optimización monoobjetivo realizada completamente.
<b>Característica:</b>	Realizar simulación monoobjetivo.
<b>Objetivo:</b>	Confirmar que se puede llevar a cabo la resolución del problema <i>PipeOptimizing</i> sobre la red abierta.
<b>Configuración:</b>	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
<b>Datos de prueba:</b>	independentRun = 10 Archivo inp: hanoi-Frankenstein.inp Archivo gama: hanoiHW.Gama
<b>Acciones de prueba:</b>	<ol style="list-style-type: none"> <li>1. Abrir JHawanetFramework</li> <li>2. Cargar archivo de red.</li> <li>3. Seleccionar el problema <i>PipeOptimizing del menú</i>.</li> <li>4. Configurar el problema usando la ventana de configuración.</li> <li>5. Realizar la optimización.</li> </ol>
<b>Resultados esperados:</b>	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

<b>Test ID:</b>	MT003
<b>Título:</b>	Optimización monoobjetivo cancelada.
<b>Característica:</b>	Realizar simulación monoobjetivo.
<b>Objetivo:</b>	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.
<b>Configuración:</b>	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
<b>Datos de prueba:</b>	Archivo inp: hanoi-Frankenstein.inp Archivo gama: hanoiHW.Gama
<b>Acciones de prueba:</b>	<ol style="list-style-type: none"> <li>1. Abrir JHawanetFramework</li> <li>2. Cargar archivo de red.</li> <li>3. Seleccionar el problema <i>PipeOptimizing del menú</i>.</li> <li>4. Configurar el problema usando la ventana de configuración.</li> <li>5. Realizar la optimización.</li> </ol>
<b>Resultados esperados:</b>	Al cancelar la búsqueda de soluciones la ventana de estado indica que la optimización a sido detenida.

**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 4.11: Especificación caso de prueba manual MT013

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

Cuadro 4.12: Especificación caso de prueba manual MT014

<b>Test ID:</b>	MT014
<b>Título:</b>	Ver resultados de la simulación hidráulica de un solo tiempo
<b>Característica:</b>	Realizar simulación con configuración de archivo de red.
<b>Objetivo:</b>	Confirmar que se puede visualizar los resultados de la simulación realizada.
<b>Configuración:</b>	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
<b>Datos de prueba:</b>	Archivo inp: hanoi-Frankenstein.inp Archivo gama: hanoiHW.Gama
<b>Acciones de prueba:</b>	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.
<b>Resultados esperados:</b>	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

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

Cuadro 4.14: Especificación caso de prueba manual MT016

<b>Test ID:</b>	MT016
<b>Título:</b>	Ver resultados de la simulación hidráulica de mas de un tiempo
<b>Característica:</b>	Realizar simulación con configuración de archivo de red.
<b>Objetivo:</b>	Confirmar que se puede visualizar los resultados de la simulación realizada.
<b>Configuración:</b>	El equipo tiene la aplicación JHawanetFramework lista para ejecutar.
<b>Datos de prueba:</b>	Archivo inp: Vanzyl.inp Archivo gama: VanzylConfiguration.json
<b>Acciones de prueba:</b>	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.
<b>Resultados esperados:</b>	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 se 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 a la unidad de análisis.

Durante la etapa de análisis, se procesan los datos presentando una serie de gráficas que ayudarán a visualizar de mejor manera la información recolectada.

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

### 5.2. Diseño del estudio de caso

En esta sección, como se menciono 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 practica 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 preliminar

**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

##### **Usuario**

- Yamisleydi Salgueiro
- Marco Alsina
- Sergio Silva

#### **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.

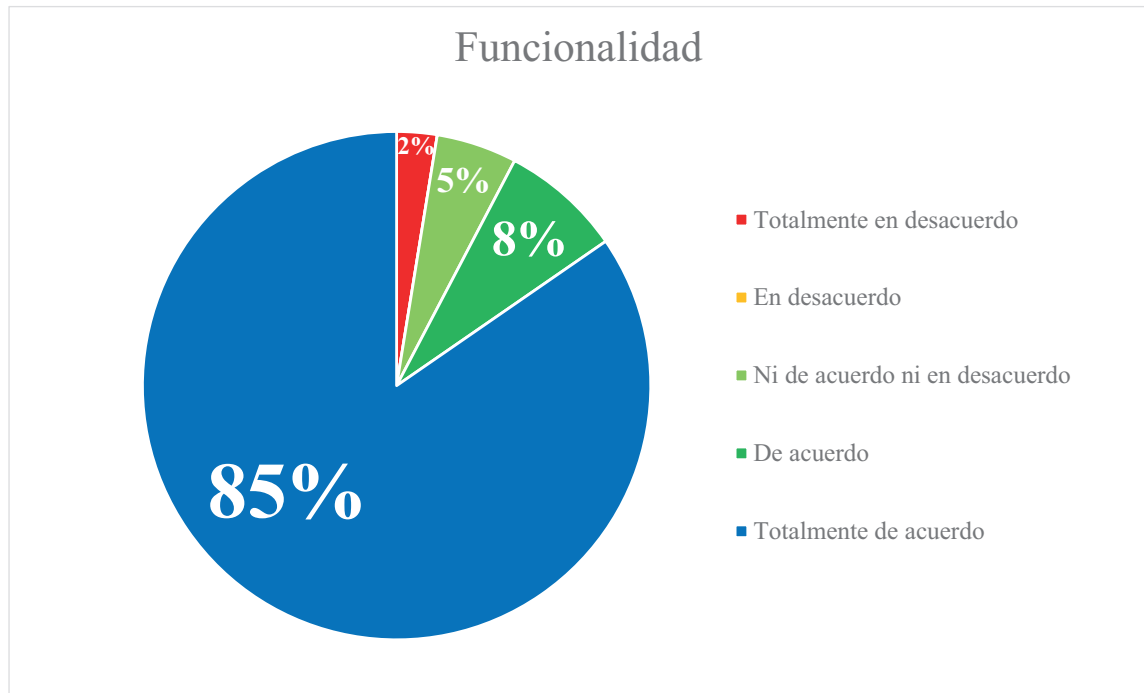


Figura 5.1: Gráfico circular de los resultados de la evaluación de funcionalidad de la aplicación

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* [32]

<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.

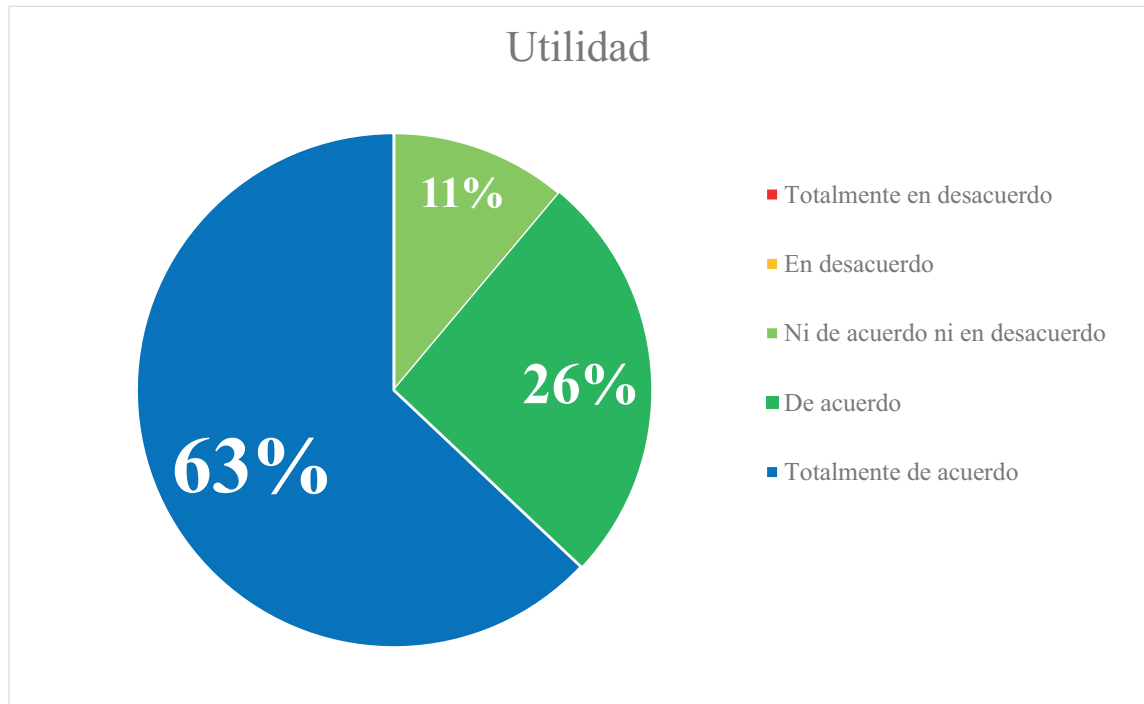


Figura 5.2: Gráfico circular de los resultados de la evaluación de utilidad de la aplicación

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 acuerdo 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.

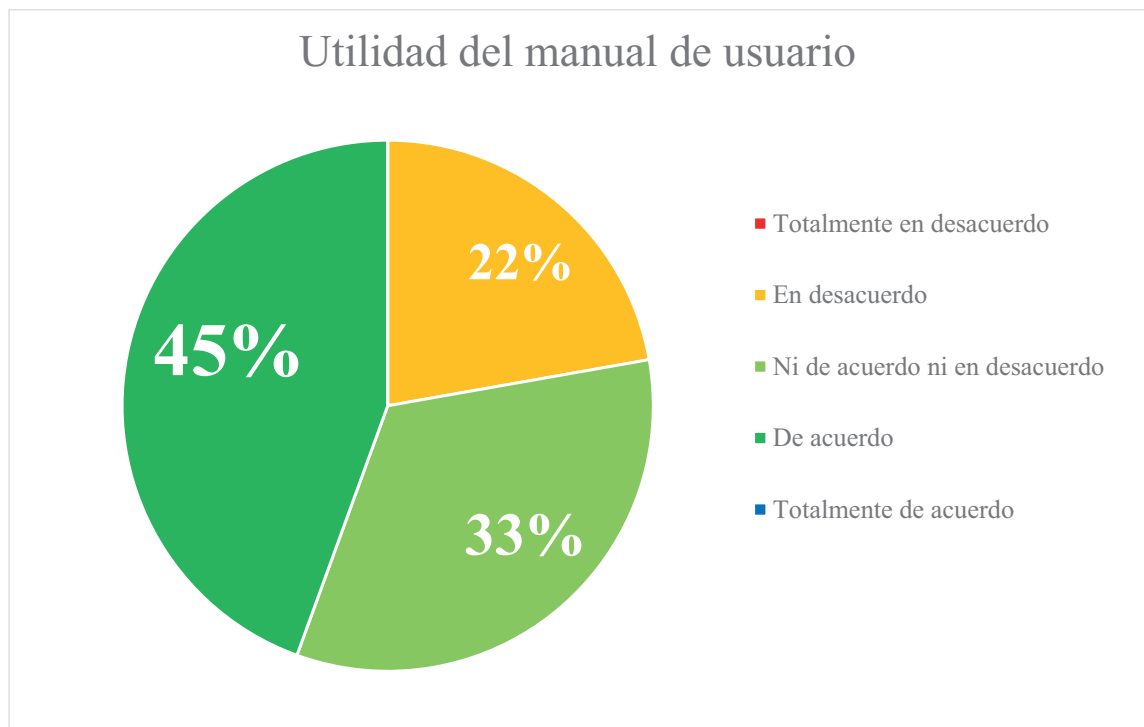


Figura 5.3: Gráfico circular de los resultados de la evaluación de la utilidad del manual de usuario

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.

## 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.

# Glosario

**RDA** Red de distribución de agua potable

**GA** Algoritmo Genético

**NSGAI** Non-Dominated Sorting Genetic Algorithm II



# Bibliografía

- [1] 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.
- [2] Donald Bell. UML basics : The component diagram. *IBM developerWorks*, (December):1–10, 2004.
- [3] Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
- [4] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.
- [5] Omid Bozorg-Haddad, Mohammad Solgi, and Hugo A Loáiciga. *Meta-heuristic and evolutionary algorithms for engineering optimization*. John Wiley & Sons, Incorporated, Newark, United States, 2017.
- [6] Mathias Braux and Jacques Noyé. Towards partially evaluating reflection in Java. *ACM SIGPLAN Notices*, 34(11):2–11, 1999.
- [7] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys (CSUR)*, 17(4):471–523, 1985.
- [8] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 2002.

- [9] 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.
- [10] 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.
- [11] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. The Java® Language Specification Java SE 8 Edition, 2015.
- [12] 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.
- [13] 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.
- [14] Dorothea Heiss-Czedik. An introduction to genetic algorithms. *Artificial Life*, 3(1):63–65, 1997.
- [15] Shrinidhi R. Hudli and Raghu V. Hudli. A Verification Strategy for Dependency Injection. *Lecture Notes on Software Engineering*, (January 2013):71–74, 2013.
- [16] Sean Luke. *Essentials of metaheuristics*. Lulu, second edition, 2013.
- [17] Yasaman Makaremi, Ali Haghighi, 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.
- [18] Robert C Martin. Iterative and incremental development ( IID ). *Design*, (Iid), 1999.

- [19] 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.
- [20] 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.
- [21] Daniel Mora. Diseño de redes de distribución de agua mediante algoritmos evolutivos. análisis de eficiencia. 2012.
- [22] 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.
- [23] Rafael Pérez Arellano and Sergio García Alcubierre. Análisis del comportamiento hidráulico de la red de abastecimiento de la ciudad de Córdoba mediante EPANET. *Jornadas de Ingeniería del Agua*, page 10, 2011.
- [24] Roger S Pressman. *Software Engineering A Practitioner’s Approach*. 7 edition, 2009.
- [25] 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.
- [26] Marc H.J. Romanycia and Francis Jeffry Pelletier. What is a heuristic? *Computational Intelligence*, 1(1):47–58, 1985.
- [27] Lewis Rossman. *EPANET 2.0 en español. Analisis hidraulico y de calidad del agua en redes de distribución de agua. manual del usuario*. 2017.
- [28] 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.
- [29] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

- [30] C L Sabharwal. Java, Java, Java. *IEEE Potentials*, 17(3):33–37, 1998.
- [31] Modeling T H E State and Laser Scanning Technology. Usage of Dependency Injection within different frameworks. (March), 2020.
- [32] Will T. Measuring and interpreting system usability scale (sus). <https://uiuxtrend.com/measuring-system-usability-scale-sus/>. Accedido 20-07-2020.
- [33] 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.
- [34] R Victor. Iterative and incremental development: A brief history. 2003.
- [35] Xin She Yang. Metaheuristic optimization. *Scholarpedia*, 6(2011):11472, 2015.

**ANEXOS**

## A. Documento de especificación de requisitos

---

## B. Documento de diseño

---

## C. Manual de usuario

---



## D. Documento de casos de prueba

---

## E. Cuestionario para la evaluación de la aplicación

---