



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

Documento de especificación de requisitos

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

EQUIPO DE DESARROLLO:

Nombre	Rol	Contacto
Gabriel Sanhueza Fuentes	Administrador, Analista, Diseñador, Implementador y Tester.	gsanhueza15@alumnos.uta.cl

CONTRAPARTE:

Nombre	Rol	Contacto
Jimmy H. Gutiérrez-Bahamondes	Cliente/Profesor guía	
Jimmy H. Gutiérrez-Bahamondes	Cliente/Profesor co-guía	

HISTORIAL DE CAMBIOS

Version	Fecha	Modificaciones
0.1	07/09/2019	Primer borrador
0.2	30/09/2019	Rellenar arquitectura lógica
0.3	05/09/2019	Agregar diagramas de clases
0.4	12/12/2019	Agregado diagrama de secuencia y módulos
0.5	13/12/2019	Agregado detalle de anotaciones
1	30/01/2020	Modificada definición de anotaciones
1.1	06/06/2020	Modificada definición de anotaciones nuevamente para satisfacer requisitos nuevos. Cambio en diseño de interfaces.

TABLA DE CONTENIDOS

	página
Tabla de Contenidos	I
Índice de Figuras	III
1. Introducción	5
1.1. Propósito del sistema	5
1.2. Alcance del proyecto	5
1.3. Definiciones, siglas y abreviaturas	6
1.4. Referencias	7
1.5. Descripción general	8
2. Diseño arquitectónico	9
2.1. Arquitectura Física	9
2.2. Arquitectura lógica	9
3. Diseño detallado	12
3.1. Diseño detallado de módulos	12
3.2. Diseño de estructura de sistema	13
3.3. Operación del sistema	16
4. Diseño de interfaces	20
4.1. Interfaz de inicio	20
4.2. Interfaz de configuración y descripción del problema	21
4.3. Interfaz de ejecución del experimento	22
4.4. Interfaz de gráficos	25
4.5. Interfaz de resultados	26
4.6. Interfaz de visualización de configuraciones de la red	27
4.7. Interfaz de ejecución de una red	28
5. Detalles de implementación	30
5.1. Uso de <i>Java Annotation</i> y <i>Java Reflection</i>	31
5.1.1. Anotaciones para los operadores	31

5.1.2. Anotaciones para los objetos que heredan la interfaz <i>Registrable</i>	32
5.2. Interfaz Registrable	39

ÍNDICE DE FIGURAS

	página
2.1. Arquitectura física.	9
2.2. Arquitectura lógica.	10
3.1. Diagrama de componentes.	12
3.2. Diagrama de clases general del sistema.	14
3.3. Diagrama de clase de la red.	15
3.4. Diagrama de clases del componente <i>metaheuristic</i>	16
3.5. Diagrama de secuencia de la carga y visualización de la red.	17
3.6. Diagrama de secuencia de la simulación de la red usando los valores del archivo de configuración de red (inp).	18
3.7. Diagrama de secuencia de la optimización.	19
4.1. Esquema de interfaz de inicio del sistema.	20
4.2. Interfaz de descripción del problema.	21
4.3. Interfaz de configuración del problema.	22
4.4. Interfaz de ejecución del experimento monoobjetivo.	23
4.5. Interfaz de ejecución del experimento multiobjetivo.	24
4.6. Esquema de interfaz para graficar las soluciones	25
4.7. Interfaz de resultados de la optimización	27
4.8. Interfaz de visualización de la configuración de la red.	28
4.9. Interfaz de resultados de ejecución de simulación hidráulica.	29
5.1. Constructor de un solo parámetro.	31
5.2. Constructor de un solo parámetro.	31
5.3. Interfaz para configurar el operador <i>UniformSelection</i>	32
5.4. Interfaz para configurar el operador <i>IntegerPolynomialMutation</i>	32
5.5. Constructor de clase que hereda de registrable y sus metadatos para cada parámetro.	33
5.6. Menú de problemas.	34
5.7. Componente <i>ComboBox</i> para configurar los operadores.	35
5.8. <i>ComboBox</i> expandido para configurar el operador.	36
5.9. Apartado para configurar el parámetro de tipo <i>File</i>	37

5.10. <i>TextField</i> presente cuando esta la anotación <i>@NumberInput</i>	37
5.11. Apartado para <i>NumberToggleInput</i> con el mismo GroupID.	38

1. Introducción

1.1. Propósito del sistema

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

1.2. Alcance del proyecto

Al final del periodo de desarrollo la herramienta contara con las siguientes prestaciones.

- El sistema permitirá la carga y la visualización de la red gráficamente.
- El sistema inicialmente solo resolverá dos clases de problemas de optimización, uno monoobjetivo y el otro multiobjetivo. El problema monoobjetivo será el de los costos de inversión. En cuanto al problema multiobjetivo, este será el de los costos energéticos y el número de encendidos y apagado de las bombas.
- El sistema inicialmente únicamente contara con dos algoritmos implementados los cuales serán el algoritmo genético y NSGA-II. El algoritmo genético será el usado para tratar el problema monoobjetivo, mientras que NSGA-II será aplicado al multiobjetivo.

Este proyecto no contempla la creación de la red por lo que estas deberán ser ingresadas como entradas al programa. La creación de las redes puede realizarse a través de EPANET. Además, esta herramienta únicamente podrá ser ocupada en equipos cuyo sistema operativo sea Windows debido a que se realizan llamadas a librerías nativas.

1.3. Definiciones, siglas y abreviaturas

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

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

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

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

Java Annotation: Característica de java para agregar metadatos a elementos de java (clases, métodos, parámetros, etc.) [7]. Las anotaciones no tienen efecto directo sobre el código, pero cuando son usadas junto con otras herramientas pueden llegar a ser muy útiles. Estas herramientas pueden analizar estas anotaciones y realizar acciones en base a estas, por ejemplo, generar archivos adicionales como clases de java, archivos XML, entre otras; ser analizadas du-

rante la ejecución del programa vía *Java Reflection*, para crear objetos cuyo tipo no conocemos en tiempo de compilación; etc.

GUI: Graphical User Interface.

1.4. Referencias

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

1.5. Descripción general

El proyecto será un medio que permitirá la realización de simulaciones hidráulicas aplicando metaheurísticas con el fin de buscar soluciones a problemas comunes en redes de distribución de agua potable.

Inicialmente se incorporará la presencia de dos metaheurísticas las cuales serán el Algoritmo Genético y el algoritmo NSGA-II. Cada uno de estos algoritmos estará enfocado en la búsqueda de soluciones para un problema determinado. El algoritmo GA estará enfocado en la búsqueda de soluciones para el objetivo de costos de inversión desde el enfoque monoobjetivo. En cambio, el algoritmo NSGA-II buscara soluciones para el problema multiobjetivo, cuyos objetivos serán el costo de operación y el régimen de bombeo (*Pumping Scheduling*).

2. Diseño arquitectónico

2.1. Arquitectura Física

La arquitectura física del software a desarrollar consistirá en la arquitectura monolítica. Puesto que la herramienta a desarrollar es un programa de escritorio cuya persistencia de datos se realiza en el mismo equipo del usuario que está ejecutando la aplicación. En la Figura 2.1 se puede ver una imagen que representa la arquitectura.

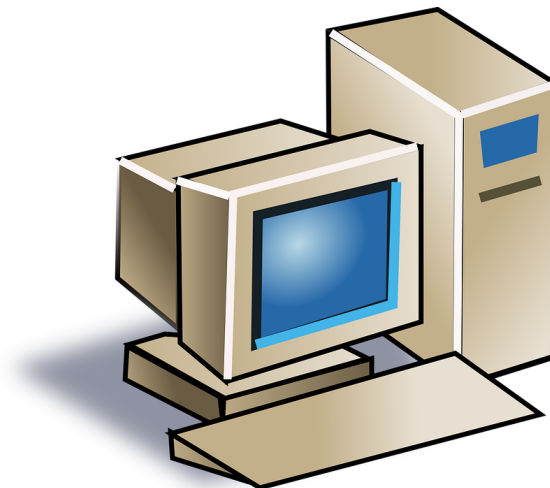


Figura 2.1: Arquitectura física.

2.2. Arquitectura lógica

La arquitectura lógica que se utilizara corresponde al patrón Modelo-Vista-Controlador [8]. La elección de este patrón se debe a que permite la separación de la lógica de negocio

y la vista presentada al usuario logrando de esta manera una aplicación altamente mantenible y con una mejor escalabilidad. El diagrama que representa la arquitectura usada para el desarrollo se puede ver en la Figura 2.2.

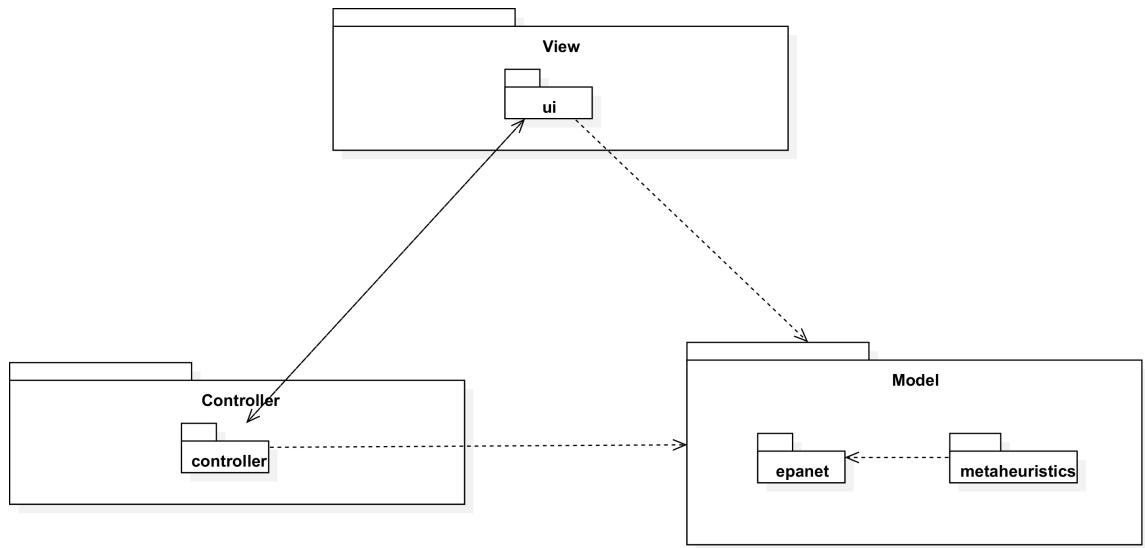


Figura 2.2: Arquitectura lógica.

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

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

Componente Metaheurísticas (*Metaheuristics*): Módulo que contiene los algoritmos, operadores, los tipos de solución permitidos y los problemas que serán abarcados durante el desarrollo del proyecto.

Componente Hidráulico (EPANET): El módulo hidráulico posee las clases necesarias para cargar los archivos de red (inp) y generar una representación de

ellos a través de una serie de clases. Este módulo también se encarga de guardar la representación de una red ya modificada en un archivo inp para que pueda ser usado por el programa EPANET. Las simulaciones hidráulicas serán realizadas usando la librería epajava. Esta librería realiza las llamadas nativas a la EpanetToolkit. La librería epajava puede ser descargada en el repositorio de git ubicado en <https://github.com/jhawanet/epajava> .

3. Diseño detallado

3.1. Diseño detallado de módulos

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

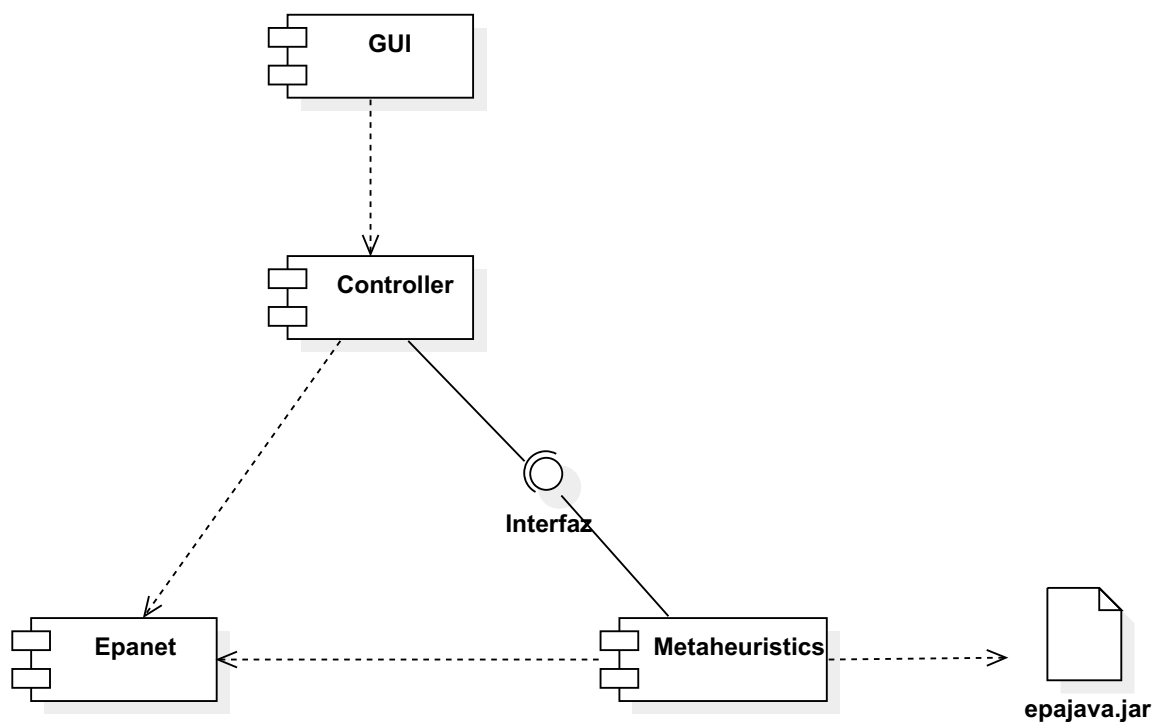


Figura 3.1: Diagrama de componentes.

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

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

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

Epanet Este componente esta encargada de la lectura y la escritura de archivos inp. Este componente cuenta también con clases que representan una red cargada desde un inp. Estas clases permitirán editar, durante la ejecución del programa, algunas configuraciones de la red con el fin de crear un nuevo archivo de descripción de la red.

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

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

3.2. Diseño de estructura de sistema

En la Ilustración 4 se muestra un diagrama general de los componentes, sus clases más importantes y su interacción. En la Figura 3.2 se muestra un diagrama general de los componentes, sus clases más importantes y su interacción.

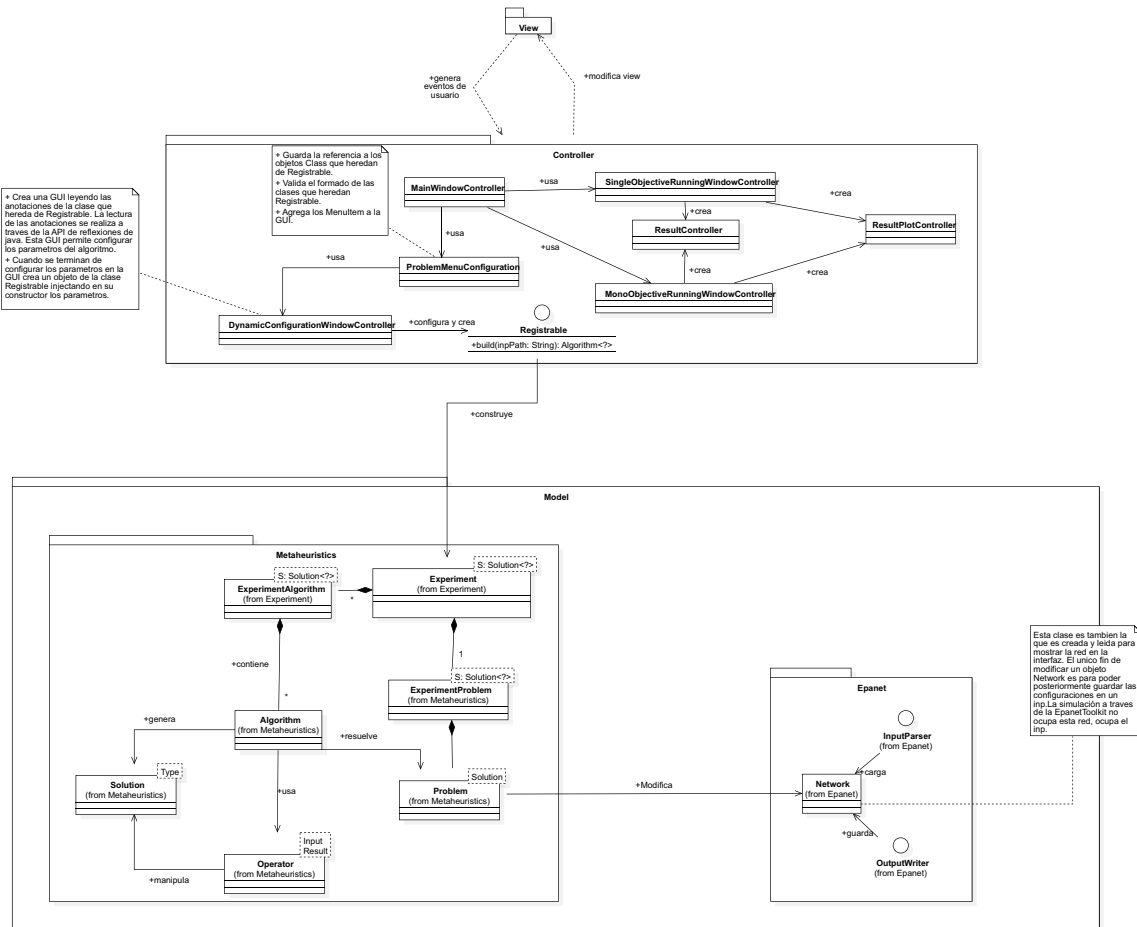


Figura 3.2: Diagrama de clases general del sistema.

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

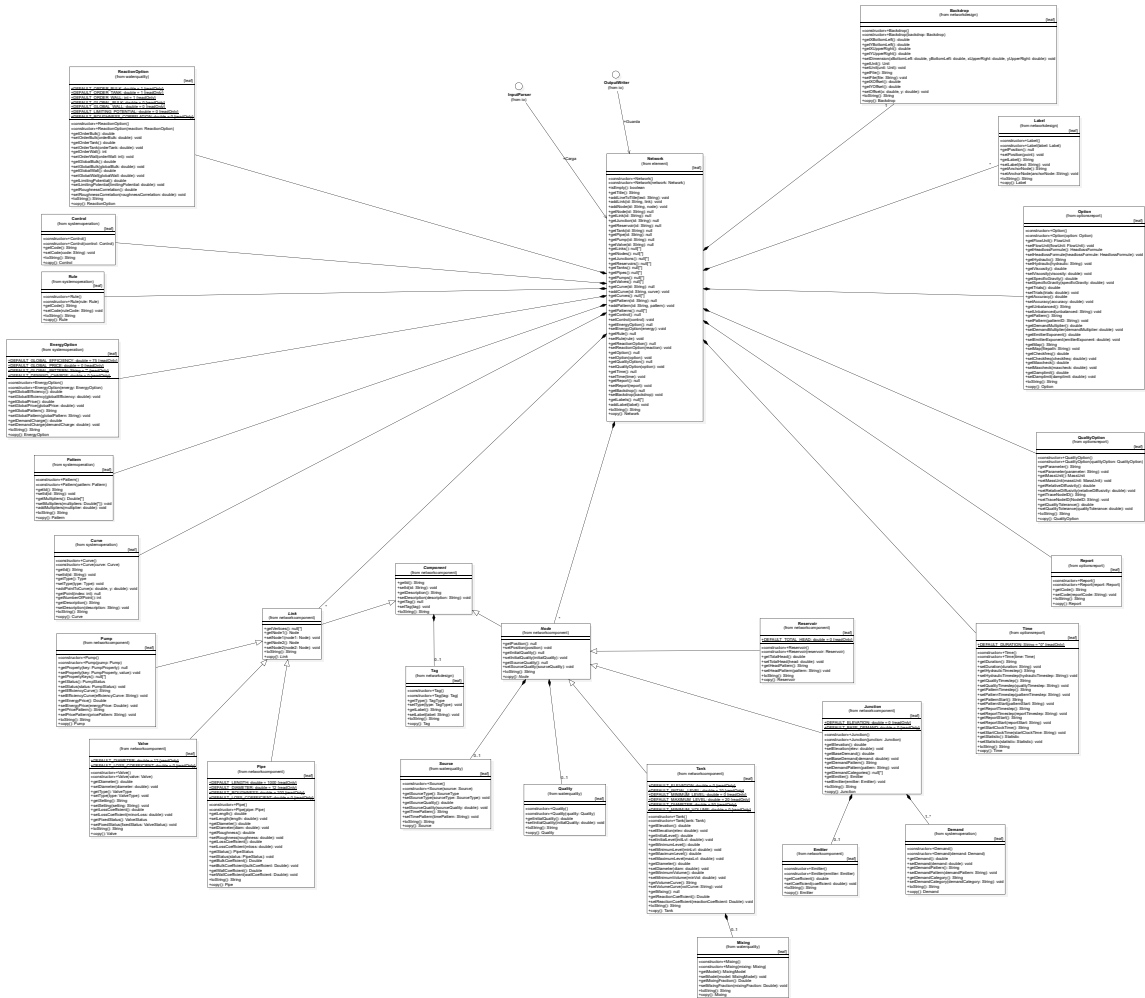
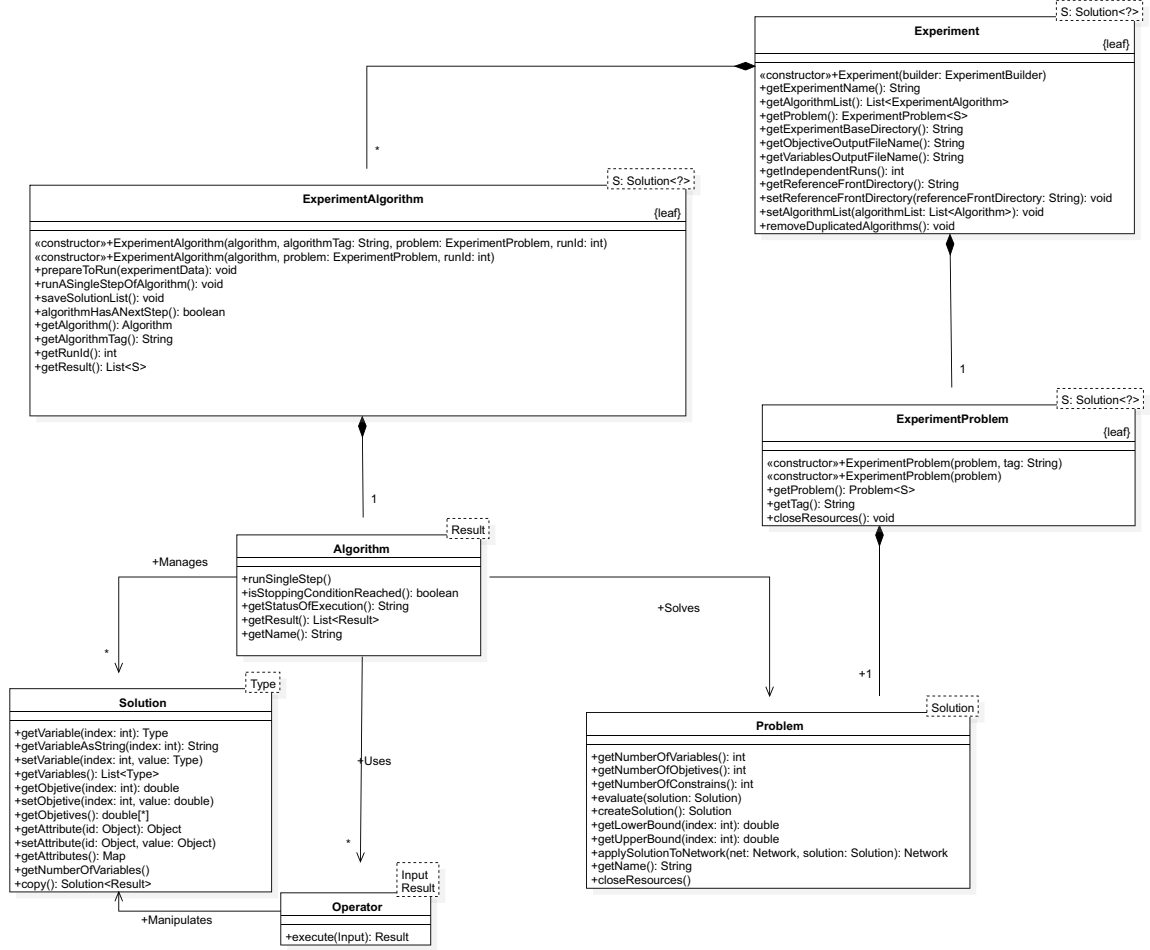


Figura 3.3: Diagrama de clase de la red.

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

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

3.3. Operación del sistema

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

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

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

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

y multiobjetivo.

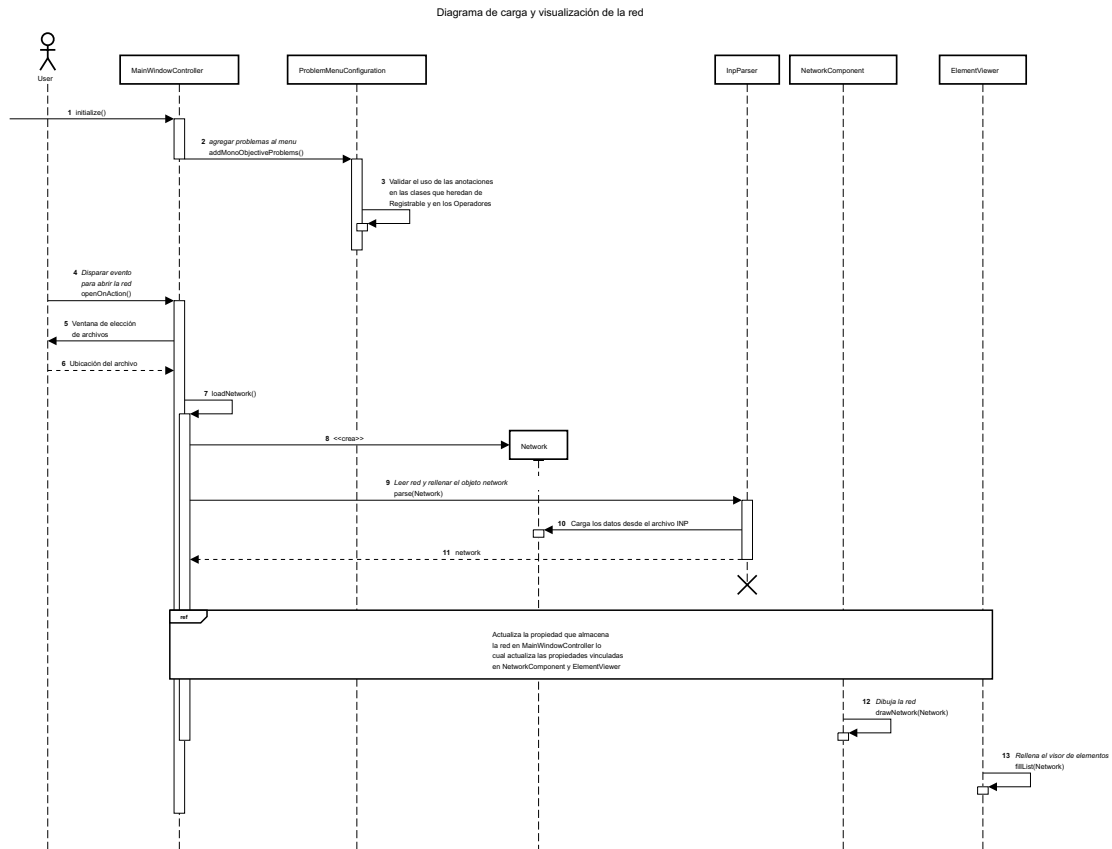


Figura 3.5: Diagrama de secuencia de la carga y visualización de la red.

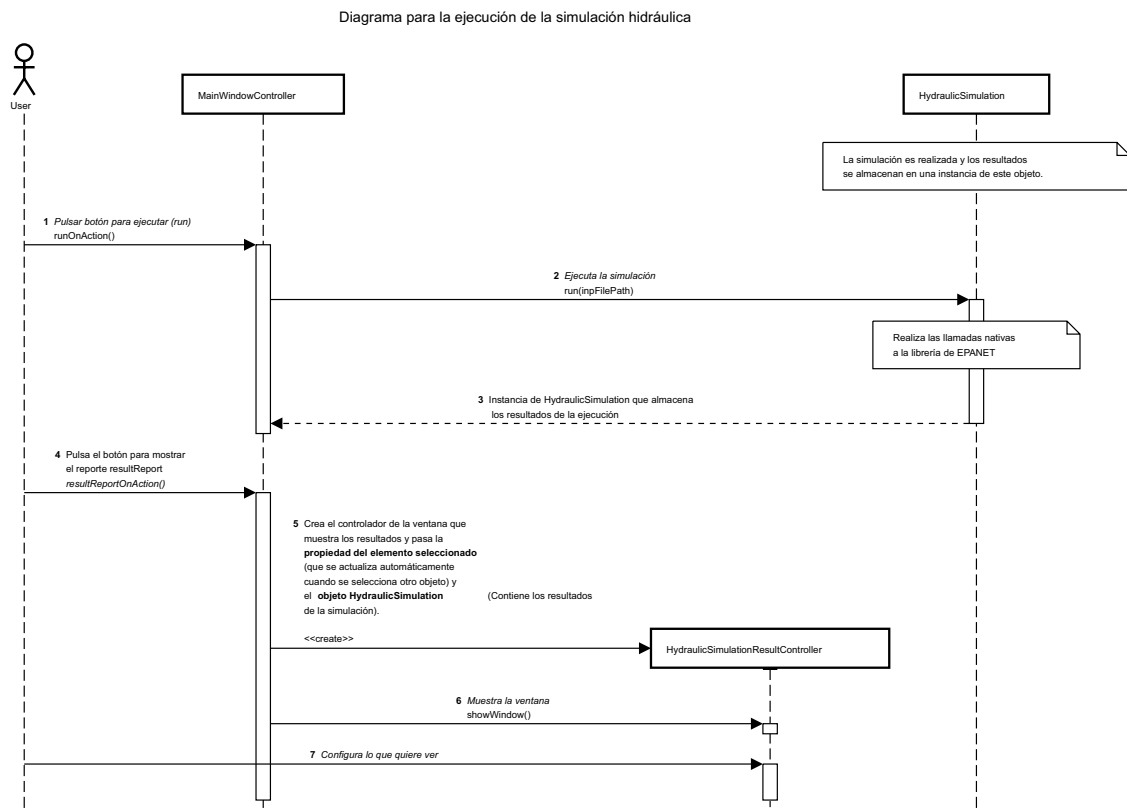


Figura 3.6: Diagrama de secuencia de la simulación de la red usando los valores del archivo de configuración de red (inp).



4. Diseño de interfaces

4.1. Interfaz de inicio

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

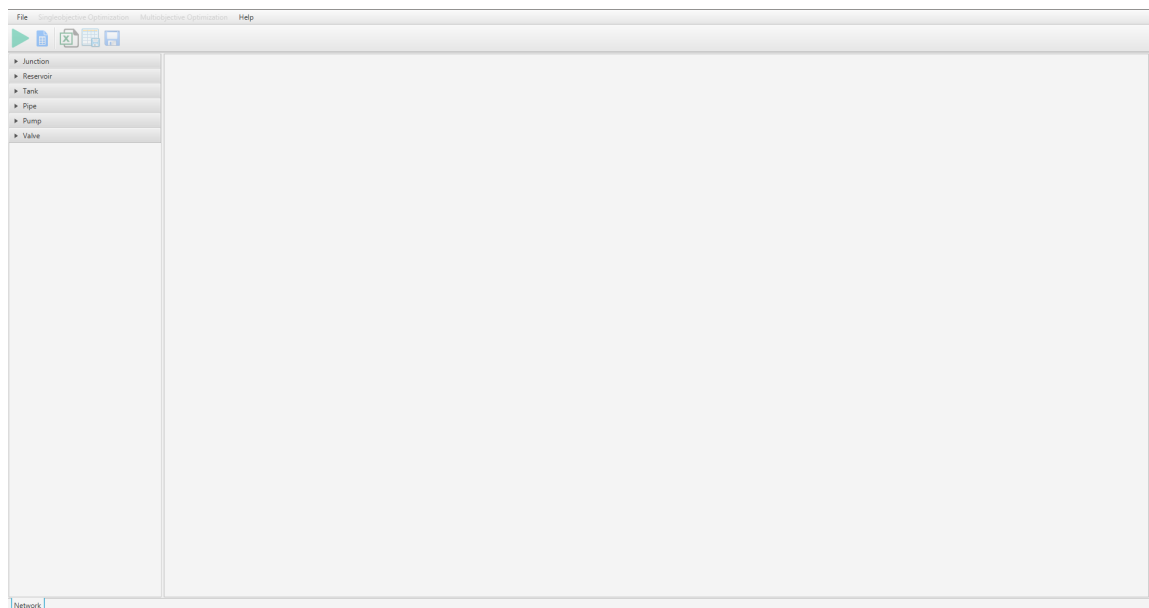


Figura 4.1: Esquema de interfaz de inicio del sistema.

4.2. Interfaz de configuración y descripción del problema

Esta interfaz es mostrada cuando el problema tiene parámetros que configurar. Se puede ver un ejemplo del formato de esta en las siguientes Figuras. La Figura 4.2 muestra la interfaz de descripción del problema a optimizar. La Figura 4.3 muestra la interfaz para configurar los parámetros del problema.

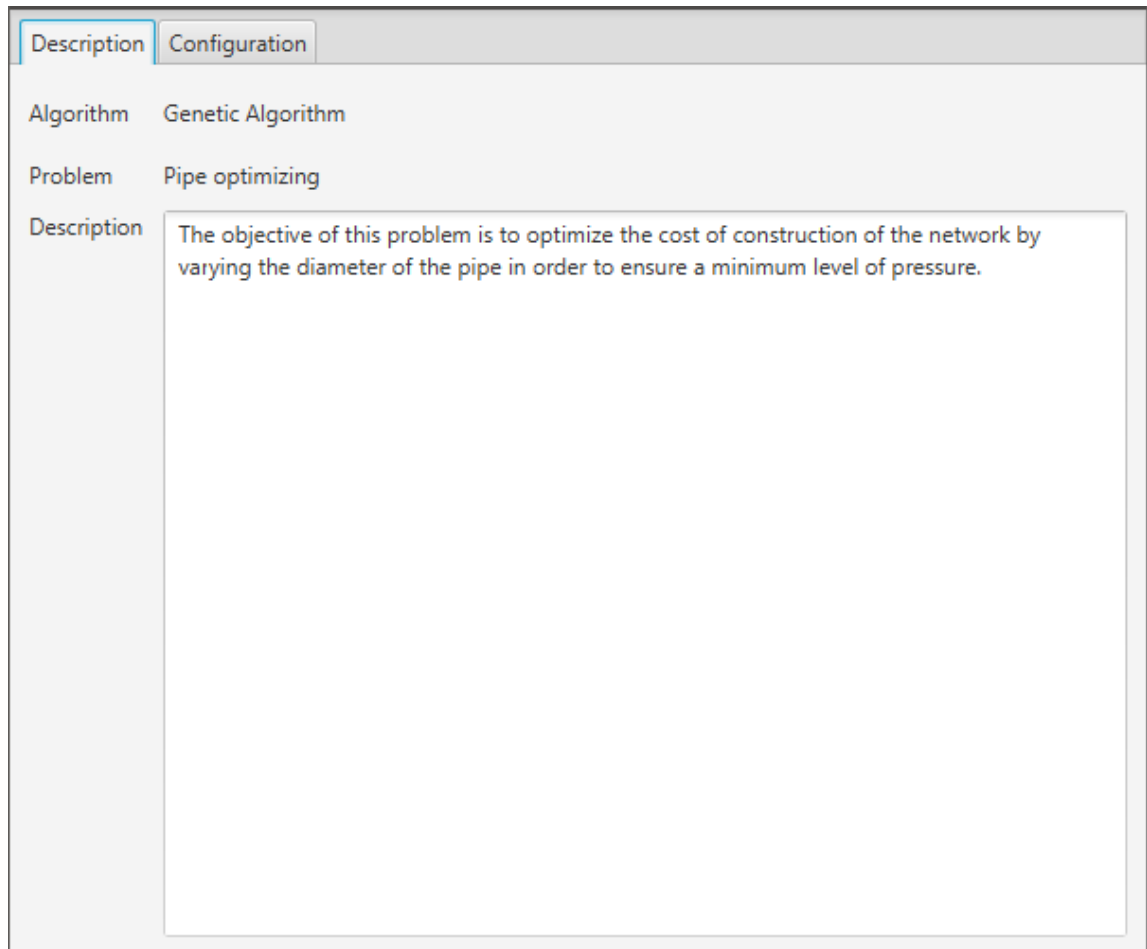


Figura 4.2: Interfaz de descripción del problema.

Description	Configuration
Independent run	5
Min pressure	30
Population Size	1000
Finish Condition	
<input checked="" type="radio"/> Number of iteration without improvement	100
<input type="radio"/> Max number of evaluation	1000
Gama	<input type="text"/> <button>Browse</button>
Selection Operator	Uniform Selection <button>Configure</button>
Crossover Operator	Integer Single Point Crossover <button>Configure</button>
Mutation Operator	Integer Simple Random Mutation <button>Configure</button>

Run Cancel

Figura 4.3: Interfaz de configuración del problema.

4.3. Interfaz de ejecución del experimento

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

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

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

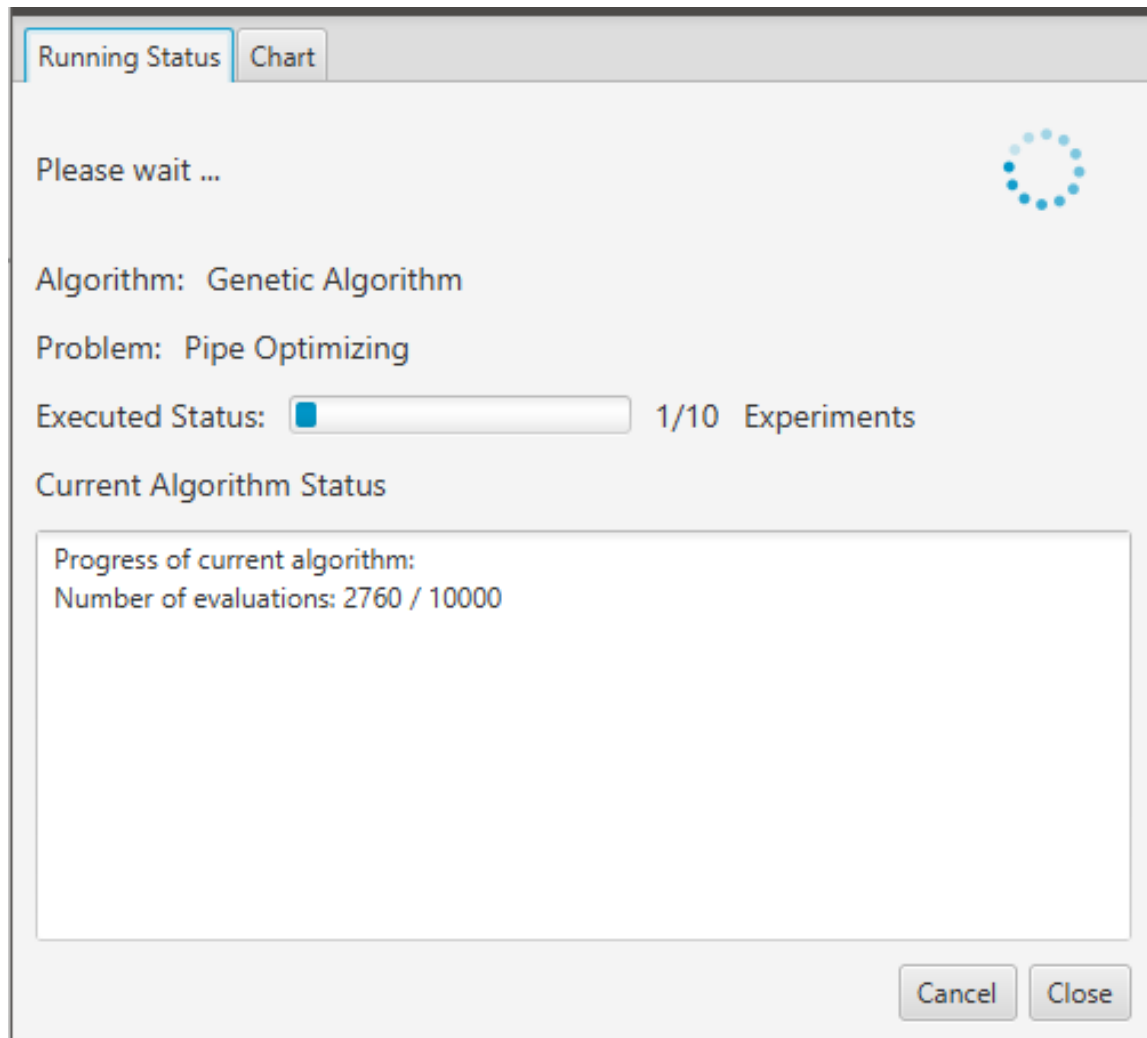


Figura 4.4: Interfaz de ejecución del experimento monoobjetivo.

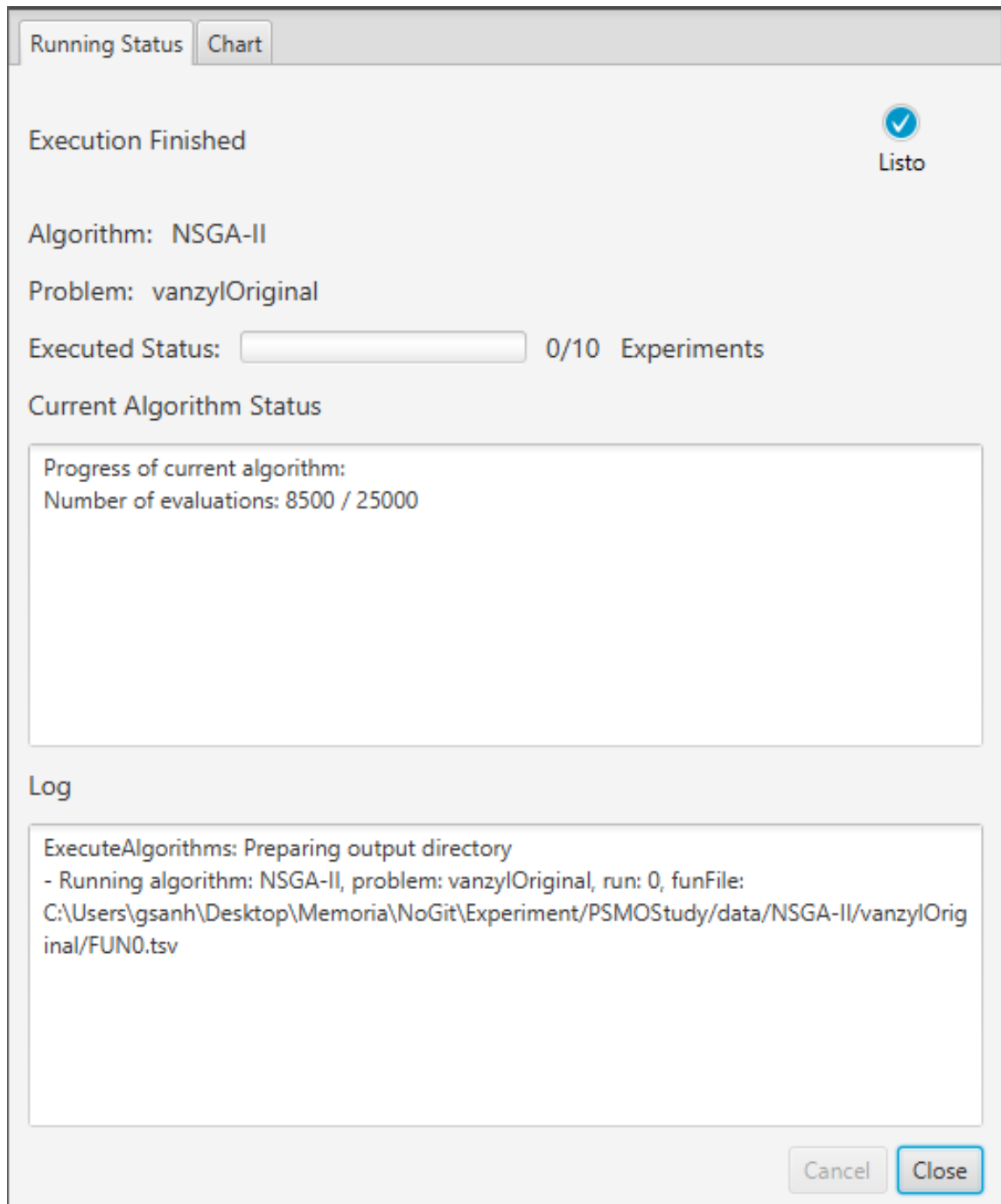


Figura 4.5: Interfaz de ejecución del experimento multiobjetivo.

4.4. Interfaz de gráficos

Esta interfaz será mostrada cuando la pestaña “*Chart*” sea seleccionada. Esta interfaz contara con un gráfico de dos ejes. Si el problema es de un objetivo, entonces el eje vertical corresponderá al valor del objetivo después de realizar la evaluación. Mientras que el eje horizontal corresponderá al número de generaciones. Si el problema tiene dos objetivos, el eje vertical corresponderá al primer objetivo y el eje horizontal corresponderá al segundo objetivo. Esta interfaz es mostrada en la Figura 4.6.

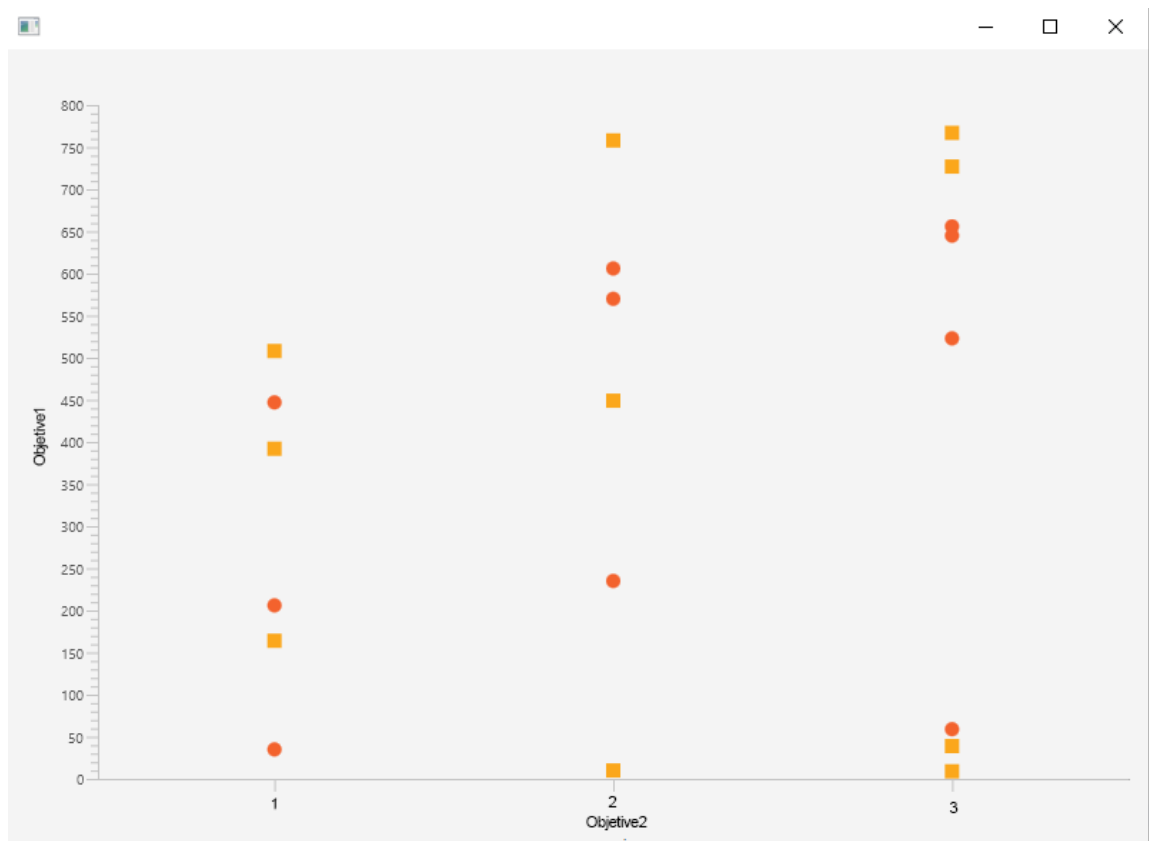


Figura 4.6: Esquema de interfaz para graficar las soluciones

4.5. Interfaz de resultados

La interfaz de resultados será mostrada cuando la ejecución del algoritmo haya terminado exitosamente. Esta interfaz es mostrada en la ventana principal de la aplicación en una nueva pestaña. Al seleccionar una pestaña de resultado se activan tres botones como se muestra en la Figura 4.7.

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

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

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

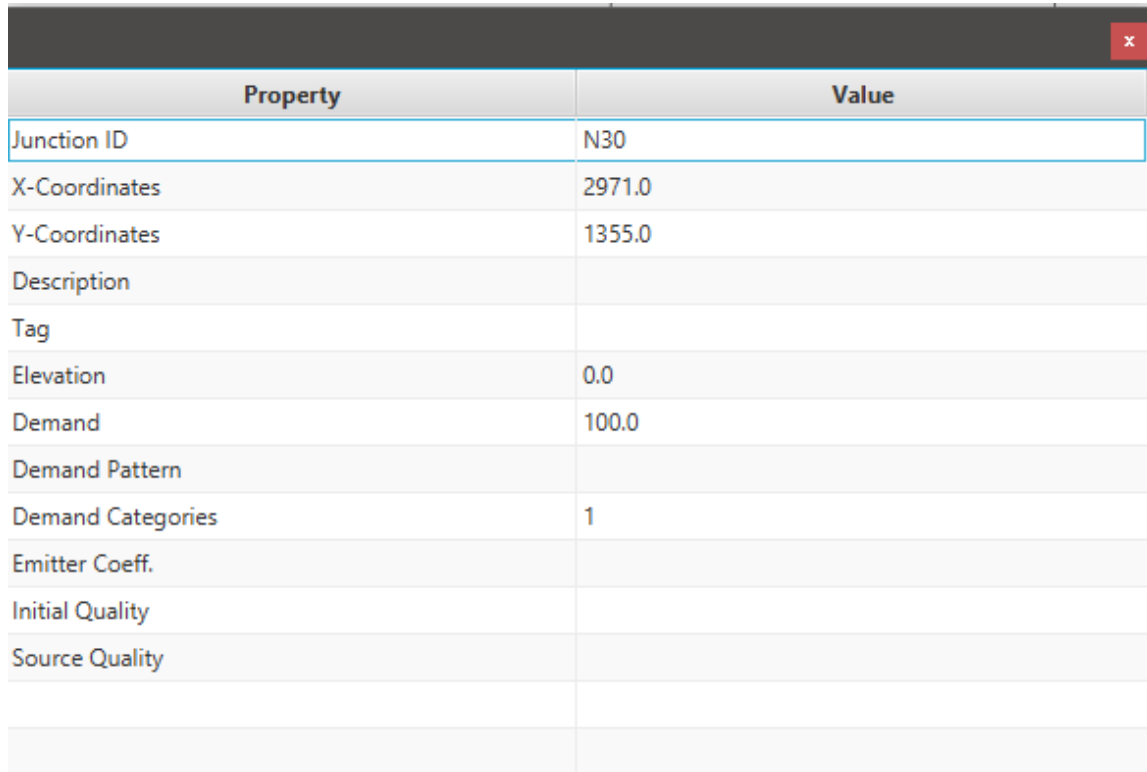
Objective 1	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X25	X26	X27	X28	X29	X30	X31	X32	X33	X34	Overall Constrains Violation	Number of Generation	Min Pressure	Population	
6700749.918494301	6	6	4	4	1	3	4	5	6	6	6	6	6	5	4	1	6	3	1	4	4	5	3	6	2	1	1	2	4	6	6	5	5	5	6	0.0	1000	30	10
66080327.25913304	6	5	5	4	5	6	4	6	6	6	6	6	6	3	1	6	4	3	1	1	3	2	5	2	1	1	1	4	6	5	5	3	2	1	0.0	1000	30	10	
6498076.728730202	6	4	5	4	5	5	6	6	6	6	6	6	6	3	1	6	5	5	2	1	1	2	5	2	1	1	1	4	5	4	3	3	1	1	0.0	1000	30	10	
6899153.985442001	6	5	4	5	4	3	4	5	5	6	6	6	6	3	1	6	4	2	3	1	1	5	6	4	3	3	4	1	6	5	6	5	5	5	0.0	1000	30	10	
6965245.5751485005	6	5	5	5	2	2	3	6	6	5	6	6	5	4	2	6	3	1	3	4	5	3	6	3	2	1	1	4	6	6	5	5	4	5	0.0	1000	30	10	
6482691.777673552	6	5	4	4	5	5	6	6	6	6	6	6	6	3	1	6	5	3	2	2	1	3	5	3	1	1	2	3	4	5	4	2	1	1	0.0	1000	30	10	
6389710.047555201	6	5	4	4	1	1	2	4	4	5	4	6	6	3	1	6	4	5	2	1	1	3	6	4	3	1	1	2	6	6	6	6	5	5	0.0	1000	30	10	
6591050.153720099	6	5	5	4	5	5	5	6	6	6	6	6	5	3	2	5	4	1	5	4	6	1	6	1	3	3	3	4	6	5	5	4	4	3	0.0	1000	30	10	
6934641.500937499	6	6	5	5	4	6	4	4	5	6	5	6	6	3	1	6	6	5	5	3	1	3	6	1	1	1	2	3	5	6	5	5	4	5	0.0	1000	30	10	
6805819.275237002	6	5	5	5	4	5	4	6	5	6	6	6	6	3	1	6	5	4	1	1	2	2	6	1	1	1	2	3	5	6	5	5	3	3	0.0	1000	30	10	

</

Figura 4.7: Interfaz de resultados de la optimización

4.6. Interfaz de visualización de configuraciones de la red

Al seleccionar un componente de la red en la interfaz gráfica y hacer doble click se abrirá una interfaz que permitirá ver la configuración por defecto de los componentes de la red. Esta interfaz consiste en mostrar una tabla en donde la primera columna será el nombre del atributo de la red y la segunda el valor. Un ejemplo de esta interfaz para un componente de la red se puede visualizar en la Figura 4.8.



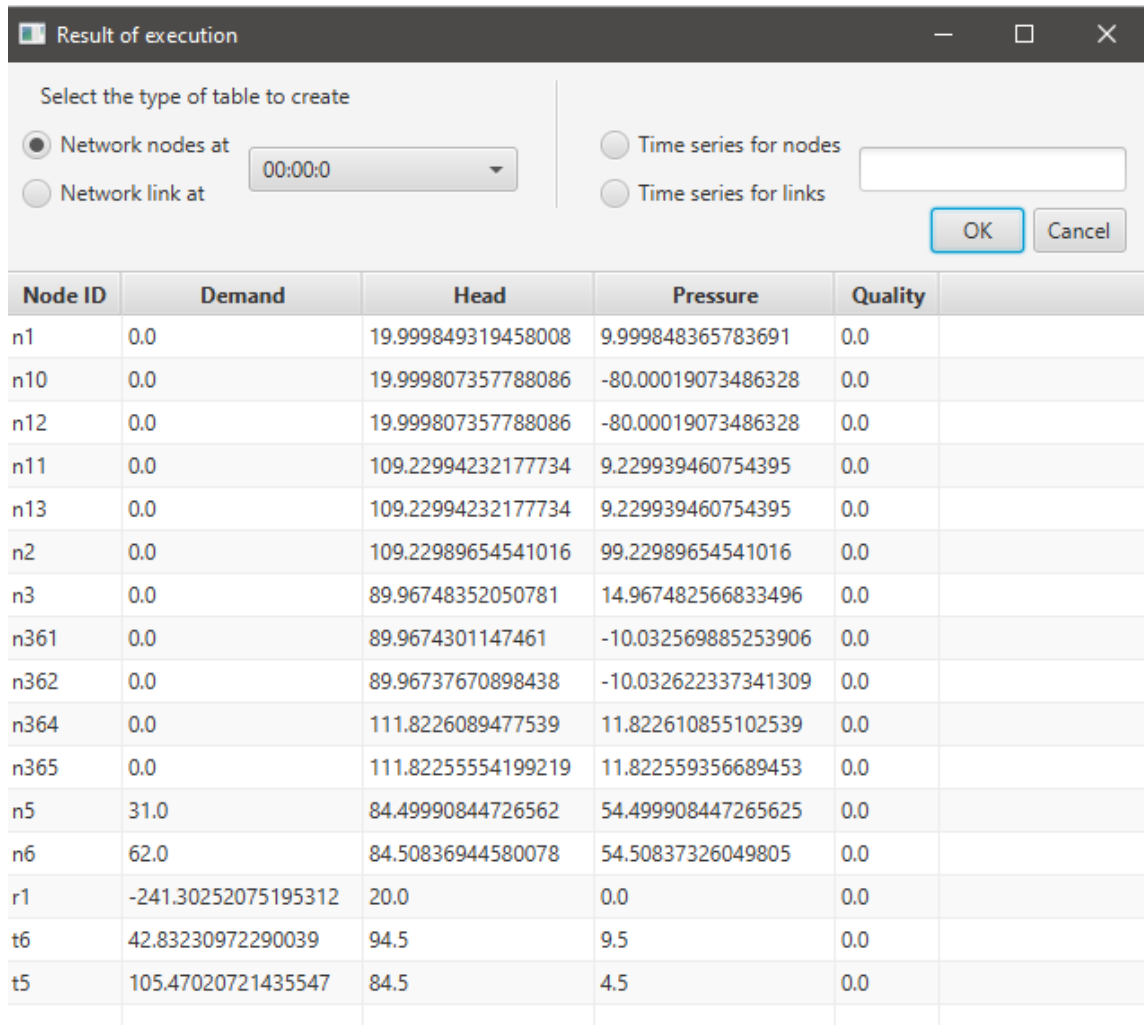
The image shows a screenshot of a software window titled "Network Configuration" (partially visible). It contains a table with two columns: "Property" and "Value". The table lists various network parameters and their current values. A red 'x' button is visible in the top right corner of the window.

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

Figura 4.8: Interfaz de visualización de la configuración de la red.

4.7. Interfaz de ejecución de una red

Se puede ejecutar una red con su configuración por defecto. Una vez ejecutada la red se puede visualizar en una interfaz los resultados de su ejecución. Este interfaz se muestra en la Figura 4.9.



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

Figura 4.9: Interfaz de resultados de ejecución de simulación hidráulica.

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

5. Detalles de implementación

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

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

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

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

este sea solicitado a través del método declarado por la interfaz. El nombre de dicha interfaz es *Registrable*.

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

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

5.1.1. Anotaciones para los operadores

@DefaultConstructor

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

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

Figura 5.1: Constructor de un solo parámetro.

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

Figura 5.2: Constructor de un solo parámetro.

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

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


A screenshot of a configuration dialog for the *UniformSelection* operator. It features a label 'constant' followed by a text input field containing the value '1.5'. Below the input field are two buttons: 'Save' and 'Cancel'.

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

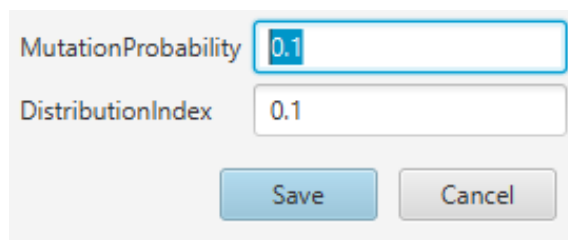
A screenshot of a configuration dialog for the *IntegerPolynomialMutation* operator. It contains two labels: 'MutationProbability' and 'DistributionIndex'. Each label is followed by a text input field; the first contains '0.1' and the second contains '0.1'. At the bottom are 'Save' and 'Cancel' buttons.

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

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

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

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

```

@NewProblem(displayName = "Pipe optimizing", algorithmName = "Genetic Algorithm",
description = "The objective of this problem is to optimize the cost of "
"construction of the network by varying the diameter of the pipe in order "
"to ensure a minimum level of pressure.")
@Parameters(operators = {
    @OperatorInput(displayName = "Selection Operator", value = {
        @OperatorOption(displayName = "Uniform Selection", value = UniformSelection.class)
    }),
    @OperatorInput(displayName = "Crossover Operator", value = {
        @OperatorOption(displayName = "Integer Single Point Crossover", value = IntegerSinglePointCrossover.class),
        @OperatorOption(displayName = "Integer 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 = 1000)
    }, //
    numbersToggle = {
        @NumberToggleInput(groupID = "Finish Condition", displayName = "Number of iteration without improvement", defaultValue = 100),
        @NumberToggleInput(groupID = "Finish Condition", displayName = "Max number of evaluation", defaultValue = 1000)
    })
public PipeOptimizingRegister(Object selectionOperator, Object crossoverOperator, Object mutationOperator, File gama, int independentRun,
..... int minPressure, int populationSize, int numberWithoutImprovement, int maxEvaluations) throws Exception

```

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

Los elementos en esta anotación consisten en:

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

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

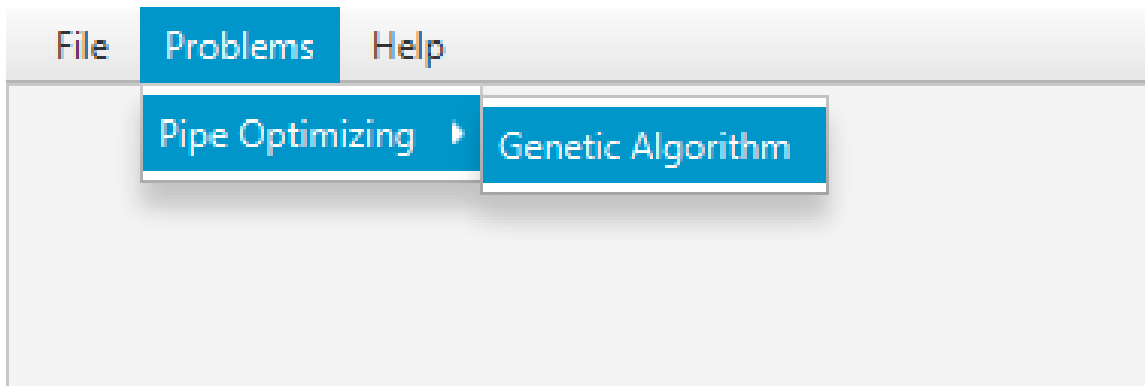


Figura 5.6: Menú de problemas.

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

@Parameters

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

1. *Object*

2. *File*

3. *int* o *double*

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

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

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

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

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

@OperatorInput

Esta anotación agrega más información a uno de los parámetros del constructor. Dentro de esta anotación existen varios elementos. Estos elementos son:

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

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

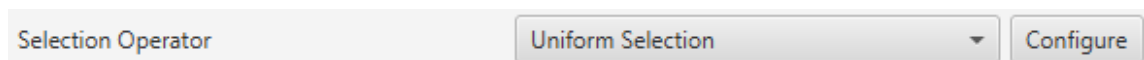


Figura 5.7: Componente *ComboBox* para configurar los operadores.

Las alternativas disponibles dentro del *ComboBox* están dadas por aquellas indicadas en el elemento *value* de este operador. Como se muestra en la Figura 5.5, la

única alternativa para el *Selection Operator* es el operador *UniformSelection* apreciado en la Figura 5.8.

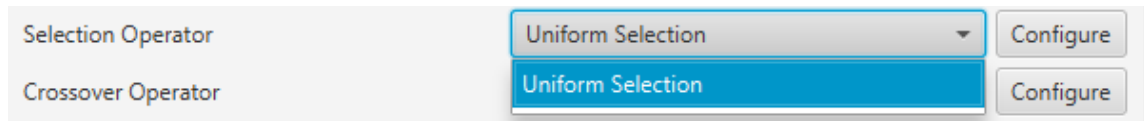


Figura 5.8: *ComboBox* expandido para configurar el operador.

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

@OperatorOption

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

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

@FileInput

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

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

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

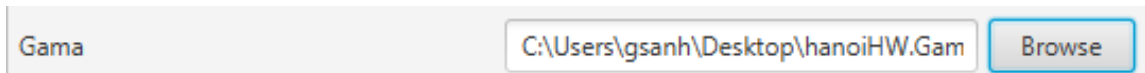


Figura 5.9: Apartado para configurar el parámetro de tipo *File*.

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

@NumberInput

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

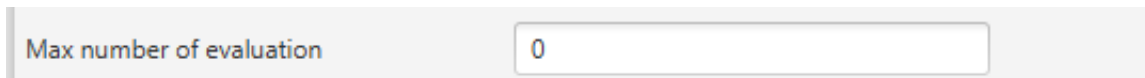


Figura 5.10: *TextField* presente cuando esta la anotación *@NumberInput*.

Dentro de esta anotación existen los siguientes elementos:

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

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

@NumberToggleInput

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

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



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

El parámetro configurado en la interfaz de usuario recibirá el valor indicado en el *TextField*. Si el *TextField* queda vacío entonces recibirá el valor cero. Sin embargo, los demás parámetros, cuyos *TextField* están deshabilitados, recibirán el valor ***Double.MIN_VALUE***, si el parámetro es de tipo *double* o *Double*; o ***Integer.MIN_VALUE*** si el parámetro es de tipo *int* o *Integer*. Por ejemplo, en la

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

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

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

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

5.2. Interfaz Registrable

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

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

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

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

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

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

Las nuevas clases deben implementar la interfaz *SingleObjectiveRegistrable* o *MultiObjectiveRegistrable*, dependiendo del tipo de problema a tratar. Las clases que implementen cualquiera de estas dos interfaces deben ser guardados en una estructura de datos, la cual será recorrida cuando se inicie la ejecución del programa y analizada usando la *Java Reflection API*. Este análisis consistirá en escanear y validar

el cumplimiento de la convención establecida para las clases que implementan esta interfaz. Esta convención consiste en lo siguiente:

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

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

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

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

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

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