EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

"POR MI PATRIA Y POR MI BIEN"

TESIS

ANÁLISIS DE DESEMPEÑO USANDO VISTHAA APLICADO A UN ALGORITMO METAHEURÍSTICO DE UN FRAMEWORK DE OPTIMIZACIÓN

Que para obtener el grado de:

Maestro en Ciencias de la Computación

Presenta:

Ing. Juan Gerardo Ponce Nájera

G12071285

Director de Tesis

Dra. Claudia Guadalupe Gómez Santillán

Co- director de Tesis

Dra. María Lucila Morales Rodríguez

Cd. Madero, Tamaulipas Mayo 2021

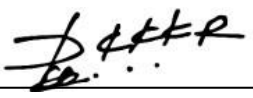
Declaración de originalidad

Yo, Juan Gerardo Ponce Nájera, en mi calidad de autor, declaro y manifiesto que este documento de tesis es producto de mi trabajo original y que no infringe los derechos de terceros, tales como derechos de publicación, derechos de autor, patentes y similares.

También, declaro que las citas textuales que he incluido (las cuales aparecen entre comillas) y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y las publicaciones. Esta tesis fue evaluada por la herramienta Turnitin obteniendo como resultado el 4% de similitud con otros trabajos.

Además, en caso de presentarse cualquier reclamación o infracción por parte de terceros en cuanto a los derechos de autor sobre la obra en cuestión, acepto toda la responsabilidad de tal infracción y relevo de ésta a mi director y codirectores de tesis, así como al Tecnológico Nacional de México y a sus respectivas autoridades.

13 de abril de 2021, Cd. Madero, Tamps.



Ing. Juan Gerardo Ponce Nájera

Agradecimientos

Agradezco a mi asesor en esta investigación, Dra. Claudia Guadalupe Gómez Santillán gracias por su trato hacia mi lleno de consejos, paciencia, apoyo y disciplina durante todo este tiempo. Mi agradecimiento más sincero para usted por el tiempo invertido sobre mí y mi proyecto, lo que me ha permitido generar un cariño, y respeto aun mayor al que poesía hacia ella en un principio.

Agradezco infinitamente a los integrantes de mi comité tutorial: Dra. María Lucila Morales Rodríguez, Dr. José Antonio Martínez Flores, Dr. Héctor Joaquín Fraire Huacuja, Dra. Laura Cruz Reyes, y el Dr. Nelson Rangel Valdez por sus sabios consejos durante mis estudios de maestría. Mi más sincero reconocimiento, no sólo por ser unos excelentes docentes e investigadores, sino también por ser un ejemplo viviente de lucha constante, trabajo y superación. Nunca olvidaré las lecciones aprendidas de ustedes.

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) y a la Dirección General de Educación Tecnológica (DGEST) por su apoyo durante todo el proceso que llevó a la conclusión de este documento, y finalmente pero no menos importante el Tecnológico Nacional de México y, sobre todo, al Instituto Tecnológico de Ciudad Madero.

Agradezco a Dios por brindarme sabiduría, inteligencia y paciencia, así como a mi familia por alentarme a seguir adelante a cada paso que daba en esta nueva etapa de mi vida, así como su compresión en el transcurso de esta. Las palabras no bastan para expresar todo lo que les debo, los que me mantienen con perspectiva, gracias.

Resumen

La comunicación es un proceso esencial para la sociedad, permite a personas, empresas y sistemas; transmitir, recibir, expresar y compartir datos e información, para realizar diferentes tareas u operaciones con esta información y reiniciar el proceso una vez más y hacer llegar a quien lo requiera.

En esta tesis se presenta la metodología de comunicación entre una herramienta de análisis de desempeño de algoritmos y un framework de optimización. La comunicación entre estas herramientas tiene el objetivo de robustecer el proceso de análisis de desempeño de algoritmos metaheurísticos.

Mediante este trabajo se aporta una metodología de comunicación genérica. La cual será utilizada para permitir desarrollar análisis de desempeño de algoritmos utilizando los métodos de la herramienta de análisis de desempeño de algoritmos usando los algoritmos desarrollados dentro del framework de optimización.

Summary

Communication is an essential process for society, it allows people, companies, and systems to transmit, receive, express, and share data and information, to perform different tasks or operations with this information and to restart the process once again and make it reach whoever requires it.

This thesis presents the communication methodology between an algorithm performance analysis tool and an optimization framework. The communication between these tools aims to strengthen the process of performance analysis of metaheuristic algorithms.

This work provides a generic communication methodology. This will be used to allow the development of performance analysis of algorithms using the methods of the algorithm performance analysis tool using the algorithms developed within the optimization framework.

Contenido

Índice de figuras

Índice de Tablas

Capítulo 1 Introducción

En esta sección se presenta el panorama general de la tesis, comenzando por la descripción de las motivaciones que dieron como resultado esta investigación, los antecedentes y el estado de arte que otorgaron información para la realización de esta. Posteriormente se explican los objetivos planteados a cumplir, además de describir los alcances y limitaciones del trabajo de investigación. Concretando con una breve descripción del contenido de cada capítulo de la tesis.

1.1 Motivaciones

En la actualidad los investigadores desean seleccionar algoritmos que solucionen de la mejor manera los problemas reales de la vida cotidiana, no existe un algoritmo que supere a otros algoritmos para todas las instancias a analizar **[Worlper97]** por lo que un factor importante en el análisis de desempeño de algoritmos metaheurísticos son la definición del problema, las instancias y las características del algoritmo.

Existen herramientas que permiten realizar el análisis de desempeño de algoritmos con base en distintos métodos, un ejemplo es VisTHAA [Castillo, 2011] una herramienta de diagnóstico de algoritmos que incluye métodos de análisis de las instancias, los algoritmos de solución y del desempeño a través de la calidad de los resultados.

VisTHAA permite incluir algoritmos de solución para realizar pruebas de análisis, pero debido a que la metodología que permite la inclusión de nuevos algoritmos metaheurísticos de solución no se encuentra trabajada del todo, al investigador le resultaría complejo el agregar sus algoritmos al repositorio interno de VisTHAA.

Debido a la dificultad que presenta el agregar algoritmos solucionadores, VisTHAA no cuenta con una gran cantidad de algoritmos metaheurísticos de solución, para sortear este problema se propone conectar VisTHAA con un framework de optimización el cual permita que VisTHAA pueda acceder a una mayor cantidad de algoritmos de una forma más sencilla.

Los frameworks según [Haro, 2008] son necesarios dado que permiten al desarrollador empezar a trabajar desde una base ya establecida, en la que él puede realizar los cambios que le sean requeridos reduciendo el tiempo de desarrollo de un algoritmo metaheurístico considerablemente.

En este trabajo se diseñó e implementó un módulo que permita la comunicación entre VisTHAA y el framework de optimización [Rojas,2017] llamado M-SDOSS. Para lograr este objetivo se incorpora un módulo de configuración para experimentos computacionales.

Los módulos que se incorporaron a VisTHAA cumplen las siguientes funciones: a) permitir generar la configuración de una experimentación y b) recabar la información ingresada de la configuración del experimento computacional y posteriormente procesan estos datos y los prepara para ser enviados al M-SDOSS el cual ejecuta la configuración del experimento con base en la información y retorna un archivo

con los resultados del experimento, una vez de vuelta en VisTHAA podrán ser llevados a cabo los procesos de análisis de algoritmos.

1.2 Antecedentes

En esta sección se presentan el conjunto de áreas de conocimiento que brindarán información sobre las herramientas y procesos que se utilizaron para la realización del trabajo final de esta tesis. Dichas áreas de investigación son:

1. Procesos de análisis de algoritmos metaheurísticos.
2. Desarrollo y configuración de algoritmos, problemas e instancias.
3. Comunicación entre diferentes sistemas.

Los trabajos presentados aquí se dividen en dos áreas de desarrollo siendo los trabajos 1, 2 y 3 los desarrollos y aportaciones realizadas a VisTHAA como herramienta de análisis, pudiendo destacar el trabajo más reciente como el primero presentado en la lista, que agrega un módulo causal a la herramienta. Y los trabajos 4 y 5, aquellos de donde se toma la gestión de procesos mediante patrones de diseño para el proceso de comunicación y el desarrollo principal del framework que fue seleccionado para conectase con VisTHAA en este trabajo de tesis, respectivamente.

1. Evaluación de estrategias de mejora de desempeño de metaheurísticas aplicados a BBP vía diagnóstico visual [García, 2019].
2. Modelado del desempeño de problemas NP-Duros aplicando análisis causal para la herramienta VisTHAA [Pérez, 2014].
3. Análisis de algoritmos metaheurísticos vía diagnóstico estadístico [Castillo, 2011].
4. Desarrollo de un protocolo de comunicación para un framework de apoyo a la toma de decisiones [Sánchez, 2016].
5. Estudio del impacto de patrones de diseño en la implementación de un framework de optimización para apoyo a la toma de decisiones [Rojas, 2017].

Estos 5 trabajos brindaron las bases para modelar y proponer el desarrollo entregado en este trabajo, permitiendo entender los funcionamientos de VisTHAA y M-SDOSS, de esta manera permitir generar un protocolo de comunicación general para futuras conexiones a distintos frameworks de optimización.

1.3 Estado del arte

En esta sección se hace mención algunos trabajos relacionados, que contribuyeron a la realización de este proyecto, específicamente aquellos trabajos relacionados con herramientas de análisis de desempeño de instancias, algoritmos y resultados. Los trabajos son presentados a continuación y en la Tabla 1 se muestra las características más relevantes de los trabajos relacionados con el análisis de desempeño de algoritmos.

- [Bisdorff, 2015], Decision Deck.
  - Es una herramienta de apoyo para la toma de decisión que funciona bajo la idea de framework tanto web como en aplicación.
  - Es un software desarrollado en código abierto que provee herramientas en una plataforma integrada por módulos interconectados.
  - Sus componentes implementan funcionalidades para el problema de toma de decisiones.
  - Su meta es proporcionar una plataforma para la toma de decisiones facilitando la prueba y comparación de los diferentes métodos integrados dentro del software.
  - Teniendo la forma de integrar más herramientas o métodos.
  - Actualmente este software fue puesto en espera al no contar con fondos para desarrollo de actualizaciones.

- [Balderas, 2012], Sistema de Apoyo a la Decisión (SAD) para SPP Framework interactivo.
  - Es un Sistema de Apoyo a la Decisión basado en métodos de interacción para la selección de carteras de proyectos públicos con información preferencial incompleta.
  - Cuenta con una arquitectura modular que permite la integración de módulos desarrollados anteriormente, así como de nuevos.
  - Usa una metodología de interacción para las etapas de captura, diseño y elección del proceso de toma de decisiones.
  - Su interfaz es intuitiva, brinda un marco de referencia al Tomar de decisiones, además de proporcionar información visual para facilitar la toma de decisiones.

- [Humeau et al, 2013], ParadisEO.
  - Arquitectura de caja blanca dedicada al diseño de metaheurísticos reusables, metaheurísticos híbridos, metaheurísticos paralelos y distribuidos.
  - Provee un amplio rango de utilidades incluyendo algoritmos evolutivos, búsquedas locales, mecanismos de hibridación, entre otros.
  - Separa los problemas que se tratan de resolver de aspectos conceptuales de los métodos de solución.
  - Permite la reutilización de código y facilita el diseño.

- [Halim et al, 2010], Visualization for analying trajectory-based metaheuristic search algorithms (VIZ).
  - Crea una herramienta de visualización con enfoque de caja blanca llamado VIZ.
  - Propone una herramienta de visualización interactiva.
  - Combina la visualización genérica aplicable en algoritmos arbitrarios con visualizaciones del problema específico.
  - Puede ser usado para analizar visualmente algoritmos de Búsqueda Local Estocástica mientras atraviesan el espacio de soluciones de los problemas de optimización combinatoria NP-duros.
  - Consiste en: a) Viz Experiment Wizard VIZ (EW); y b) Viz Single Instance Multiple Runs Analyzer (SIMRA).

- [Alcala et al, 2014], Knowledge Extraction based on Ecolutionary Learning (KEEL).
  - Es una herramienta software para evaluar algoritmos evolutivos para problemas de minería de datos incluyendo regresión, clasificación, agrupamiento, patrones de minería entre otros.

- [Casillo, 2011], Herramienta de Visualización de Algoritmos Heurísticos (VisTHAA), ver capítulo 2 sección 4.

Tabla 1 Trabajos relacionados con el análisis de desempeño de algoritmos

Autor	Problemas que resuelve	Algoritmos solucinadores	Formato	Código	se comunica con otras herramientas
Bisdorff (2015)	Variaciones de problemas de optimización con el agregado de tomador de decisiones	De la literatura	XMCDA: Un XML estandarizado para representar objetos y estructuras de datos provenientes de los diferentes algoritmos permitiendo la interacción entre ellos	Abierto	no
Balderas (2012)	SPP	-	Variante XML	Propio	no
Humeau et al (2013)	TSP,ZDT,DTLZ, Reglas de asociación, selección de variables, flowshop,biMDVRP, DVRP_MultiSwarn,WFG,UCP	Algoritmos evolutivos, búsquedas locales y mecanismos de hibridación	Variante XML	comercial	no
Halim et al (2010)	Problemas de optimización combinatoria NP-duros	Algoritmos de búsqueda local estocástica	Variante XML	-	no
Alcala et al (2014)	Problemas de minería de datos (DM) que inluyen regresión, clasificación. Agrupamiento, mineria de patrones, etc	Algoritmos evolutivos	Variante XML	Abierto	no
Casillo (2011)	Knapsack 0/1,Bin Packing classic, VCSBPP, PSP	Algoritmos heuristicos y metaheuristicos	JSON	Propio	si

Como se puede ver en la Tabla 1 el formato seleccionado por la mayoría de las herramientas son variantes no especificadas de XML, a diferencia de VisTHAA que desde antes del módulo de comunicación ya utilizaba el formato JSON para realizar algunos proceso de intercambio de información internamente, además cabe resaltar el hecho de que todas las herramientas cuentan con aplicaciones de escritorio, a excepción de Decision Deck que además presenta la posibilidad de realizar análisis vía web presentando un área de oportunidad a futuro. Por último, pero no menos importante el hecho de que ninguna de las

herramientas tiene una conexión con una herramienta (Framework de optimización) a excepción del desarrollado para esta tesis.

1.4 Justificación

El análisis de algoritmos es un área importante que da apoyo en la mejora de la calidad de las soluciones. Los algoritmos pueden ser rediseñados después de su análisis con el objetivo de mejorar la calidad y obtener el resultado óptimo. En la actualidad existen herramientas de análisis de algoritmos que están formadas por indicadores que miden la calidad de un número reducido de estrategias algorítmicas [Castillo, 2011].

Los investigadores hacen uso de las herramientas de análisis de algoritmos para mejorar la calidad de los algoritmos y brindar mejores soluciones a los problemas que se enfrentan. En la actualidad han surgido una gran variedad de estrategias algorítmicas las cuales necesitan ser programadas por los investigadores para después ser evaluadas.

Los frameworks de optimización los cuales están enfocados en resolver problemas de optimización a través de algoritmos que ya están incluidos en su núcleo. Los frameworks son herramientas que permiten el desarrollo de nuevos algoritmos y la configuración de los que tienen ya establecidos de una manera ágil por lo que permiten al investigador realizar experimentos computacionales complejos con muchísima más velocidad que en desarrollos que se producen desde cero.

Actualmente la herramienta VisTHAA no cuenta con una gran cantidad de algoritmos dentro de sí, existen distintas alternativas para lograr que VisTHAA acceda a algoritmos solucionadores que hayan sido desarrollados por frameworks de optimización, por ejemplo, la incorporación de algún framework dentro de la arquitectura de VisTHAA, o el desarrollo de un módulo que permita la conexión entre VisTHAA y algún framework de optimización.

En este trabajo se plantea la alternativa que requiere el desarrollo del módulo entre VisTHAA y un framework de optimización de tal manera que el módulo este lo más generalizado posible de esta manera estandarizando el proceso que se requiera utilizar para futuros framework y de esa manera permitiendo a VisTHAA acceder a una gran cantidad de algoritmos metaheurísticos.

1.5 Objetivos

A continuación, se presentan el objetivo general y los objetivos específicos planteados para este trabajo de investigación.

1.5.1 Objetivo general

Diseñar e implementar un módulo de comunicación entre VisTHAA y el framework de optimización M-SDOSS, integrándolas como una herramienta de análisis de algoritmos competitiva.

1.5.2 Objetivos específicos

- Analizar y seleccionar del framework de optimización.
- Desarrollar del módulo de comunicación entre VisTHAA y M-SDOSS.
- Evaluar de resultados de casos de estudio en VisTHAA.

1.6 Alcances y limitaciones

En esta sección se describen los alcances y limitaciones que forman parte de este trabajo de investigación.

1. Se trabajará la conexión entre VisTHAA y framework de optimización M-SDOSS.
2. Se añadirá un módulo que asista en la configuración de los experimentos computacionales.
3. Se modificará los módulos de caracterización de rendimiento mono-objetivo y multi-objetivo para permitir la comunicación y procesamiento de los resultados para hacer uso de los indicadores de cada módulo.
4. Se trabajará con los siguientes casos de estudio: algoritmo genético solucionando el problema de la mochila, y el NSGA-II solucionando el problema de cartera de proyectos y problemas de DTLZ.

1.7 Organización del documento

En esta sección se presenta una breve descripción de los capítulos que conforman esta tesis.

El capítulo 2 se enfoca en los fundamentos y conceptos teóricos que darán facilidad al entendimiento de esta tesis, entre ellos, los conceptos y métodos de un protocolo de comunicación, los formatos para intercambio de información, para la comunicación entre las herramientas M-SDOSS y VisTHAA.

En el capítulo 3 se explica la metodología utilizada para el cumplimiento de los objetivos específicos mencionados en la sección 1.5.2, entre ellos están la metodología usada para definir el asistente de configuración de experimentos computacionales, así como la metodología de conexión entre VisTHAA y M-SDOSS.

En el capítulo 4 se muestran los resultados obtenidos de las experimentaciones realizadas de la metodología de comunicación propuestas para los puntos citados en el capítulo 3, y la implementación e incorporación de comunicación entre a las herramientas VisTHAA y M-SDOSS para el análisis de resultados.

El capítulo 5 reporta las conclusiones obtenidas de la investigación realizada, y de igual manera se presentan las sugerencias para trabajos futuros.

El apéndice A se presenta toda la información relacionada con la librería diseñada para contener la manipulación JSON y la definición de los objetos del experimento computacional para VisTHAA.

En el apéndice B se realizó el análisis sobre el algoritmo genético implementado para VisTHAA y clases relacionadas.

En el apéndice C se realizó el análisis sobre el NSGA-II/NOSGA implementado para VisTHAA y clases relacionadas.

Capítulo 2 Marco teórico

En este capítulo se presenta una revisión de conceptos y enfoques esenciales relacionados con el desarrollo de este trabajo de investigación. Se describen el proceso de un protocolo de comunicación, se dará una introducción de las herramientas M-SDOSS (framework) y VisTHAA (análisis), además de explicarse los casos de estudio seleccionados para demostrar la comunicación entre las dos herramientas.

2.1 Introducción

En los trabajos revisados relacionados con los procesos de comunicación y el análisis de desempeño de algoritmos, se enfocan en las conexiones realizadas dentro de servicios web y trabajar sobre problemas y algoritmos definidos dentro de la misma herramienta en lugar de permitir el uso de otras herramientas especializadas en el desarrollo de algoritmos. Sin embargo, esto limita los experimentos computacionales realizables e incluso en algunos casos las configuraciones que se puede realizar sobre un algoritmo en concreto.

El traspaso de información requiere de un formato que unifique la entrada y salida de los datos para que sea entendible por los diferentes subsistemas que conformen el sistema general, además de esto, debe existir un preprocesamiento de las entradas a los diferentes subsistemas para que cada proceso interno que sea requerido no se vea afectado muy indistintamente del proceso de comunicación.

Para realizar un proceso de comunicación es necesario definir las partes implicadas en la comunicación, el formato de la información y los diferentes procesos que se requiere que sean realizados al momento de realizar esta comunicación.

2.2 Protocolo de comunicación

Dentro del ámbito de la informática es conocido como la suma de procedimientos y reglas útiles para el traspaso de datos, que es conocido por las partes implicadas en esta actividad [Diccionario L.E., 2020].

La finalidad de este mecanismo es la transmisión de datos y lograr la comunicación de distintos procesos de tal manera que se comparta una misma estructura y así como un mismo lenguaje. La literatura los protocolos son mayormente usados en redes informáticas, pero algunos frameworks han adaptado su uso para la comunicación entre distintos módulos.

Esta comunicación se logra al momento de unificar los conceptos y estructuras que se usan dentro de los diferentes módulos reduciendo la complejidad entre la interacción de los datos, haciendo que una misma entrada de datos sea estandarizada para diversos modelos compatibles, y que la representación de las soluciones usando una estructura genérica en un interfaz. Esta información estará representada bajo el protocolo de comunicación.

Por definición un protocolo son un conjunto de reglas para la representación, señalización, autentificación y detección de errores para el intercambio de información entre procesos [Davies et al., 1979], donde participan de la siguiente manera:

- Son implementados vía procesos.
    - Un proceso se ejecuta en un procesador virtual o lógico.
    - Un proceso se integra de los componentes necesarios para su operación.
    - El protocolo no es consciente, que un procesador real comparte sus recursos entre varios procesos activos.
- Entrada a los procesos ocurre por puertas lógicas de software, por donde el proceso recibe mensajes desde procesos residentes en el mismo o en otro procesador.
- Un conjunto de datos privados define el estado actual de un proceso y determinan la acción a tomar por el receptor de un mensaje.
- El resultado de la computación ejecutada por el proceso se envía por una puerta lógica de salida.

Los procesos son encontrados en:

- Equipos de una red.
- Sistema multiprocesador, para controlar interacción de procesos paralelos.
- Aplicaciones en tiempo real para el control de dispositivos.
- En cualquier sistema donde no existe relación fija en el tiempo de ocurrencia de los eventos.

Funciones de un protocolo de comunicación:

Control de errores
Protege integridad de los datos del usuario y de los mensajes de control.

Control de flujo y congestión
Permite a la red compartir sus recursos entre un gran número de usuarios, entregando a cada uno un servicio satisfactorio sin que sus operaciones corran peligro.

Control de congestión
Permite a la red compartir sus recursos entre un gran número de usuarios, entregando a cada uno un servicio satisfactorio sin que sus operaciones corran peligro.

Estrategias de encaminamiento
Permite optimizar la utilización de los recursos de la red, aumentando la disponibilidad de los servicios de la red al proveer caminos alternativos entre nodos terminales.

Los protocolos pueden ser clasificados como:

- **Directo**. Los datos e información de control pasan directamente entre las entidades sin intervención de un agente activo.
- **Indirecto**. Las dos entidades no se pueden comunicar directamente sino a través de una red conmutada o de una interconexión de redes.
- **Monolítico**. El protocolo no está estructurado en capas. El paquete debe incluir toda la lógica del protocolo.
- **Estructurado**. El protocolo posee una estructura jerárquica, en capas. Entidades de nivel inferior ofrecen servicio a entidades de nivel superior. A todo el conjunto de hardware y software, se le denomina *arquitectura*.
- **Simétrico**. La comunicación se realiza entre unidades paritarias.
- **Asimétrico**. Las entidades que se conectan no son paritarias. Por ejemplo, un proceso "cliente" y otro "servidor", o para simplificar al máximo la lógica de una de las dos entidades, de forma que una asuma la operación (Por ejemplo, en HDCL).
- **Estándares**. El protocolo es extensivo a todas las fuentes y receptores de información.
- **No estándares**. Protocolo particular. Se utiliza para situaciones de comunicación muy específicas.

Como se muestra en la Figura 1, las capacidades de un protocolo que define [Sánchez, 2007] de la siguiente manera:

- Un proceso recibe un mensaje lo procesa y envía una respuesta, sin que exista relación entre este evento y otro anterior o posterior.
- El proceso origen, conocerá la dirección del proceso destino y la incluirá en el mensaje.
- Esta dirección, identificará únicamente a un procesador, quién conocerá al proceso destino.
- El emisor envía un mensaje, entra en un estado de espera de respuesta en una de sus puertas.
- El proceso destino ejecuta la función especificada en el mensaje, construye la respuesta (con resultados y dirección del origen) y envía el mensaje respuesta por una puerta de salida, (quedando libre para aceptar otro mensaje).

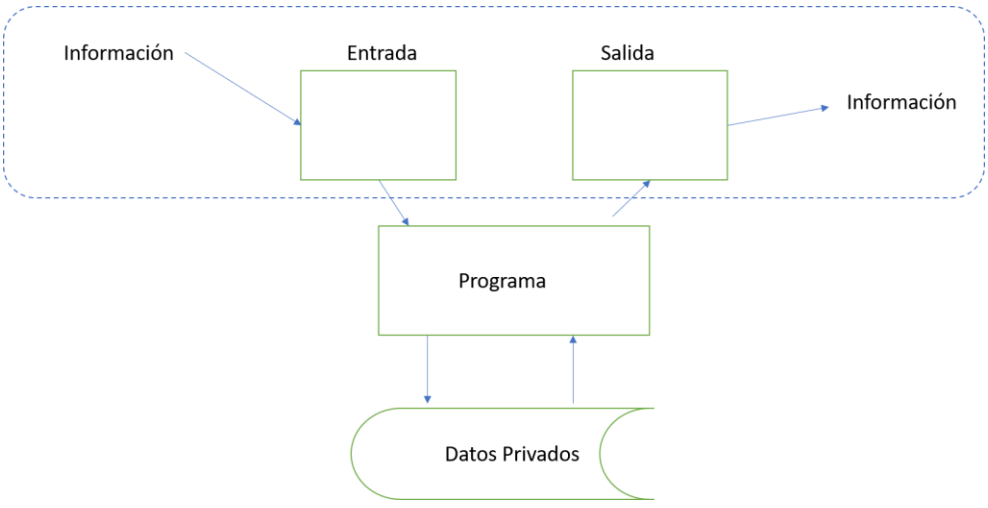


Figura 1 Diagrama de participación en el protocolo de comunicación en un proceso

- La respuesta llega al emisor, quien realiza un chequeo para asegurarse que viene del lugar correcto antes de aceptarla, luego, pasa al estado "no espera respuesta" en esa puerta de entrada.

2.3 Intercambio de información por medio de formato

En el mundo moderno los servicios, aplicaciones y los sistemas que comparten información necesitan formatos sólidos; que les permitan realizar movimientos de información compleja de manera interna. La información pudiera incluir estructuras jerárquicas y atributos variables, pero es necesario que los datos tengan una estructura de lenguaje natural para su la comprensión humana. Es en aquí donde los protocolos de comunicación permiten el paso y codificación de la información.

Algunos tipos de formatos son:

- Texto plano
- XML
- JSON
- YAML

A continuación, se describirán cada uno de los formatos mencionados.

2.3.1 Texto plano

Conocidos también como archivos de texto sin formato, en la totalidad de los casos los archivos con extensión txt, contienen información solo de texto y no provee una opción clara para informar al ordenador el lenguaje que contiene [Walsh, 1994]. En la Figura 2 se muestra un ejemplo de una descripción de un libro en texto plano.

```
Juan Ponce
Esto es un ejemplo
ejemplos
```

Figura 2 Descripción de un libro en un texto plano

2.3.2 XML

Su nombre por sus siglas en ingles Extencible Markup Language. Está compuesto de prólogo y cuerpo del documento, el cual contiene solo un elemento raíz, pero puede contener varios elementos con o sin contenido y los cuales pueden tener atributos.

Los documentos de texto, hojas de cálculo, páginas web y bases de datos son algunos de los campos de aplicación del XML. El metalenguaje aparece como un estándar que estructura el intercambio de

información varias plataformas. XML es la evolución lo que se conoce como lenguaje GML por IBM [Walsh, 1998].

Este formato es utilizado ampliamente en informática sobre todo en desarrollo de sistemas de información: caso particular a mencionar son los sistemas administrativos donde el paso de información entre módulos es importante, y se necesita una estandarización para lograr esta trasferencia, sin embargo el formato XML puede ser usado en otras áreas, como en optimización; donde es posible hacer uso de la estandarización para comunicar los resultados entre distintos módulos u otras aplicaciones para que puedan acceder a la información.
La Figura 3 presenta el ejemplo de la descripción del libro en XML.

```
<catalogo>
        <libro>
                <autor>Juan ponce</autor>
                <titulo>Esto es un ejemplo</titulo>
                <genero>ejemplos</genero>
        <libro>
<catalogo>
```

Figura 3 Descripción de un libro en formato XML

2.3.3 JSON

Es un estándar abierto que utiliza texto plano para codificar información en la estructura atributo: valor. Su nombre proviene de las ingles JavaScript Object Notation y aunque en sus primeros pasos fue considerado parte de JavaScript, JSON es independiente del lenguaje de programación y se puede encontrar disponible para los lenguajes más populares. Un archivo JSON es datos agrupados en conjuntos. Puede tener una estructura jerárquica, pero lo únicos tipos de valores son objetos, vectores, variables y valores [JSON, 2021].
En la Figura 4 se muestra el ejemplo de la descripción del libro en JSON.

```
{
Libro:{
        autor:Juan Ponce,
        titulo:Esto es un ejemplo,
        genero:ejemplos
        }
}
```

Figura 4 descripción de un libro en formato JSON

2.3.4 YAML

Este formato de intercambio de información tiene como objetivo facilitar el mapeo de estructuras de datos más complejas (como listas y arreglos asociativos) en un documento de texto plano entendible para los seres humanos. Es un formato relativamente nuevo, que se ha hecho destacar entre otros formatos más conocidos como XML y JSON debido a su característica [Sánchez, 2021].

Es a diferencia de los otros formatos más estricto y simple, por lo que se nota elegante y claro, convirtiéndolo en una excelente opción para tareas que involucren intervención humana.
En la Figura 5 se encuentra el ejemplo de la descripción del libro en YAML.

```
Libro:
        autor:Juan Ponce,
        titulo:Esto es un ejemplo,
        genero:ejemplos
```

Figura 5 descripción de un libro en formato JSON

En la Tabla 2 realizada se puede visualizar una comparativa entre todos los formatos.

Tabla 2 Comparativa entre formatos para intercambio de información [Sánchez, 2016]

Formato	Año	Legibles	Tipo de ficheros	Unicode	Tipo de estructuras	Metadatos	Lenguaje con mayor soporte	Validación
Texto plano	-	No completa	Texto	No	Ninguno	No	Todos	No
XML	1998	Si	Ficheros de cualquier formato	Si	Jérarquica	Si	Java, C#	Si
JSON	2000	Si	Texto	Si, y tiene variaciones	Objetos y arreglos	No	javaScript, Python, Java	Si
YAML	2001	Si	Texto	Si	Jérarquica	Si	Ruby on Rails	Si

Además de esto [Sánchez, 2016] en su análisis presenta la capacidad de todos los formatos por tener interoperabilidad, y por último presentando brevemente las siguientes ventajas y desventajas de cada formato.

Texto plano

Sus principales ventajas son su fácil implementación, su uso tanto para archivos de configuración, como para intercambio de datos. Entre sus desventajas encontramos que no existe un formato estándar para ser presentado por lo que la presentación de los datos variaciones de un sistema a otro pueden ser muy grandes.

XML

Este formato permite crear esquemas para fines específicos y crear nuevos tipos de etiquetas de ser requeridos, además de ser usados para bases de protocolos, servicios web, API's Rest e intercambio de datos. Por otra parte, es un formato con verbosidad, complejo, y con la finalidad de procesar documentos grandes se requiere en gran manera del uso del procesador (CPU).

JSON

Su fuerte el trabajo de estructuras animadas, su manipulación resulta sencilla para distintos lenguajes, principalmente utilizado en servicios web, API's Rest, así como el intercambio de datos. Tiene en su contra su pobre sintaxis y esto acarrea que solo pocos tipos de datos son compatibles.

YAML

Para sus pros están su simplicidad y compresión, su amplio uso en archivos de configuración, conceptos y reglas. Para sus contras está el hecho de que su uso es exclusivo para configuraciones y derivados por lo que no se presenta en muchos más ámbitos.

2.4 Herramienta de Visualización de Algoritmos Heurísticos

Este apartado se presentará la herramienta sobre la que se realizará el trabajo, su estructura interna y la funcionalidad de los módulos con los que cuenta actualmente. VisTHAA es una herramienta de diagnóstico para el análisis de algoritmos heurísticos, esto hace posible que los investigadores mejorar las capacidades funcionales mediante módulos que se integran a la arquitectura para favorecer el análisis de algoritmos heurísticos. La herramienta cuenta con los siguientes módulos disponibles [Castillo, 2011]:

- Entrada de datos y preprocesamiento
- Caracterización de instancias
- Visualización de las instancias y el comportamiento del algoritmo
- Visualización del espacio de búsqueda en tres dimensiones y el análisis de los algoritmos
- Rediseño causal

En la Figura 6 se muestra la arquitectura interna de la herramienta VisTHAA, y los módulos que tienen una dependencia con otros módulos.

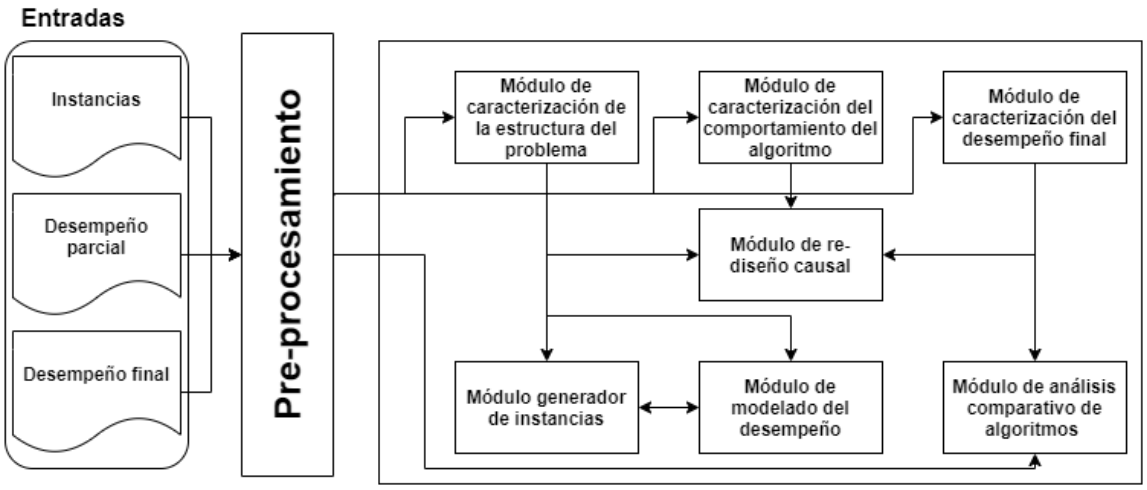


Figura 6 Arquitectura Interna de VisTHAA [García, 2019].

2.4.1 Entrada de Datos y Preprocesamiento

El módulo de entrada y preprocesamiento tiene como función que los investigadores utilicen sus propios formatos para la lectura y procesamiento de archivos, para cumplir esta función es necesario ajustar los archivos de entrada a un formato de entrada requerido por la herramienta.

El formato de instancias es descrito en un archivo (sin formato), esta descripción es conocida como metainstance. Posteriormente, el número de instancias a ser introducidos es especifico, el nombre del archivo que describe las instancias y el nombre de cada una de ellas, esta especificación recibe el nombre de logbook. En la Figura 7 se muestra la ruta para el precargado del conjunto de instancias a trabajar y en la Figura 8 se muestra los archivos principales para el proceso de carga de instancias.

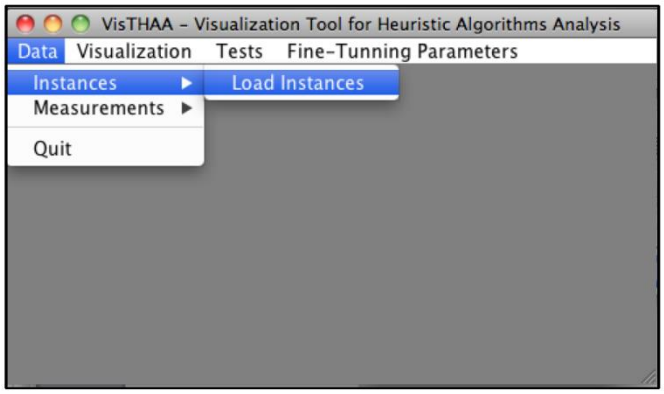
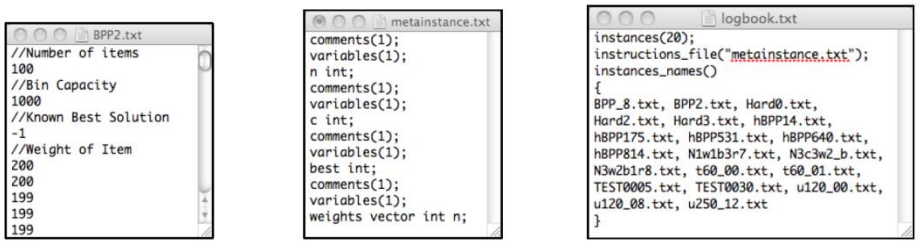


Figura 7 Ruta de Acceso para la Carga de Instancias [Castillo, 2011].



a) Instancia de BPP **b)** archivo metainstance **c)** archivo logbook

Figura 8 Visualización de los Elementos de la Entrada de Datos [Castillo, 2011].

2.4.2 Caracterización de Instancias

Una vez finalizado el proceso de carga de información, el investigador puede aplicar un proceso de caracterización al conjunto de instancias del problema a analizar. Dicho proceso puede realizarse de manera estadística o visual. La caracterización estadística se realiza a través de la aplicación de índices al conjunto de instancias, el cual tiene como objetivo caracterizar y cuantificar factores que definen la estructura del problema, desempeño parcial y final del algoritmo, permitiendo la identificación de los factores que impactan en el proceso de optimización. La caracterización visual permite mostrar de manera gráfica al investigador, los resultados obtenidos de los índices aplicados en el proceso de caracterización, permitiendo una mejor comprensión de estos, y una comparación entre los resultados obtenidos y los que fueron mejores.

2.4.2.1 Módulo de Introducción de Nuevos Índices

VisTHAA tiene implementados algunos de los índices especializados para BPP, los cuales pueden ser utilizados para obtener conocimiento de las instancias. Sin embargo, la herramienta ofrece a los

investigadores la oportunidad de introducir sus propios índices a partir de las variables definidas en el archivo metainstance.

A manera de ejemplo, el índice que mide la capacidad del contenedor ocupada por un objeto promedio es introducido, el cual a su vez será utilizado en este estudio, en la Ecuación 1 se define el índice y en la Figura 9 se muestra una imagen de cómo debe de ser introducido a la herramienta.

$$t = \frac{\sum_{i=1}^{n} \frac{w_i}{n}}{c} \quad (1)$$

Figura 9 Módulo de Introducción de Nuevos Índices [Castillo, 2011].

2.4.2.2 Módulo de Generación de Matriz de Características

Una vez que un conjunto de índices es introducido, el investigador puede decidir cuáles de ellos considerar para generar la matriz de características sobre el conjunto de instancias seleccionadas, el propósito es obtener información para el análisis de los valores calculados sobre dicho conjunto de instancias. En la Figura 10 se muestra el módulo que permite generar la matriz de características.

La matriz de características es una aportación importante en el trabajo de Castillo, ya que permite generar una matriz de datos, los cuales posteriormente pueden ser tratados con técnicas de análisis multivariado [Castillo, 2011]. La matriz de características está conformada en sus filas por las unidades experimentales (instancias) y en sus columnas por las variables / atributos / características (índices que el investigador ingrese a VisTHAA).

La ventana para realizar los ajustes tiene en su lado izquierdo, un cuadro rojo, dentro del cual aparecen las instancias previamente cargadas a la herramienta, de las cuales el investigador puede seleccionar aquellas que desee incluir en la matriz.

En la parte central de la ventana, aparece un cuadro en color azul, dentro del cual el investigador puede seleccionar los índices que desee incluir en la matriz.

Finalmente, en el lado izquierdo de la ventana, aparecen dos opciones, las cuales tienen que ver con el archivo de salida de la matriz de características. Si se selecciona “Headers” el archivo de salida incluirá el nombre de los índices, si se selecciona “Instances names”, el archivo de salida incluirá los nombres de las unidades experimentales (instancias). Además, se encuentra un botón denominado “Calculate”, el cual genera la matriz de características y el archivo de salida en formato de texto plano, así como también muestra los resultados en otra ventana.

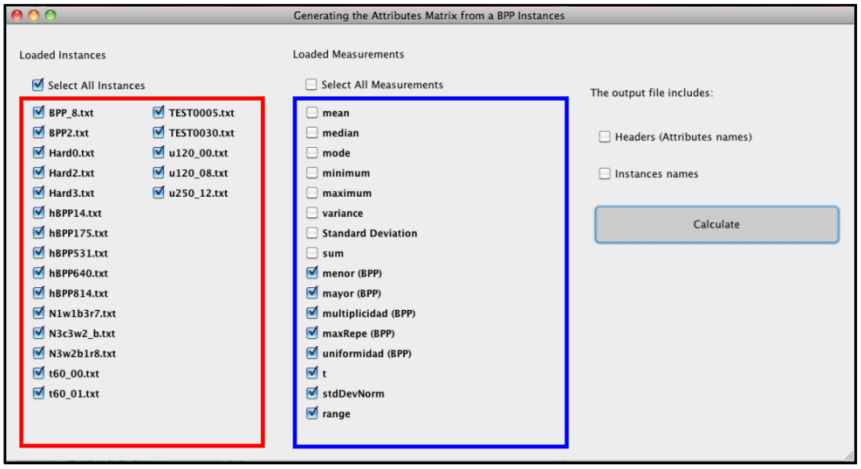


Figura 10 Módulo de Generación de Matriz de Características [Castillo, 2011].

2.4.2.3 Módulo de Calculadora

Otra contribución que está estrechamente vinculada al módulo de caracterización es el uso de una calculadora, la cual muestra variables iniciales y mediciones de propósito general. Dicho módulo permite al investigador introducir una ecuación y obtener su resultado, sin que esta ecuación se conserve en la herramienta. En la Figura 11 se muestra un ejemplo de forma de calcular un índice nuevo.

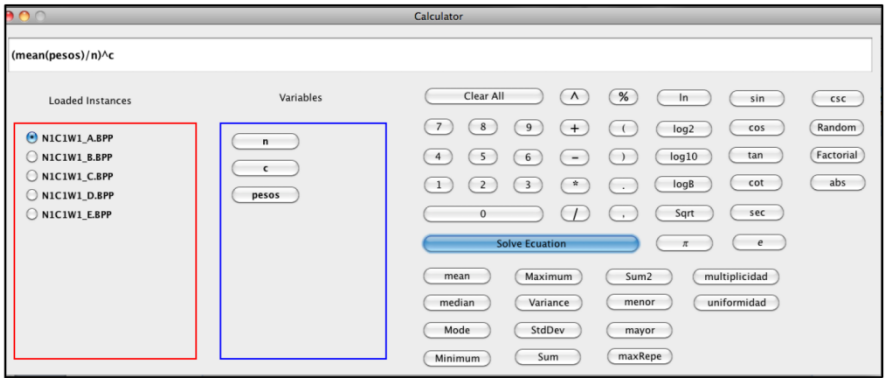


Figura 11 Ejemplo de Cálculo de un Índice Nuevo [Castillo, 2011].

De la información almacenada el investigador se le puede permitir la introducción de nuevas mediciones en notación infija, además dicho módulo utiliza el algoritmo de Shunting Yard [Dijkstra, 1961].

2.4.3 Visualización

El módulo de visualización tiene como objetivo el poder visualizar de manera gráfica las instancias cargadas en la herramienta, graficas de frecuencia de las instancias, graficas del paisaje de aptitud del comportamiento del algoritmo y graficas estadísticas.

2.4.3.1 Instancias del Problema

La visualización de la información de las instancias del problema a estudiar se realiza a través de una gráfica de barras bidimensional. En dicha estrategia, la visualización de las instancias se realiza a través de barras, donde el eje de las abscisas representa al i-ésimo objeto, mientras que el eje de las ordenadas representa el peso de los objetos. En la Figura 12 se muestra la ruta del proceso de visualización en VisTHAA.

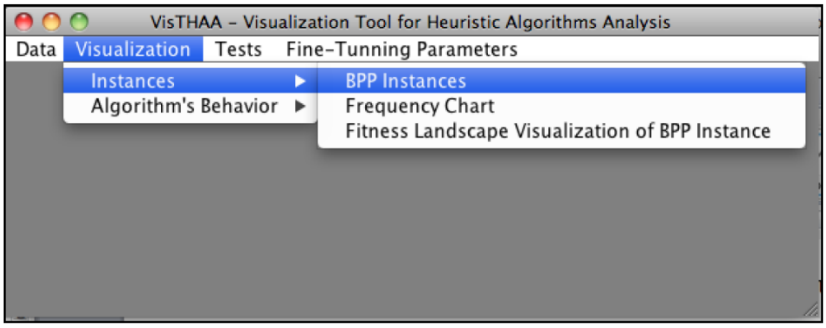


Figura 12 Ruta del Módulo de Visualización de Instancias [Castillo, 2011].

Al seleccionar el módulo de visualización de instancias, aparecerá una ventana en la cual se deberá seleccionar una de las instancias cargadas, con el fin de visualizar la información de la instancia o también mostrar la información de la instancia en forma ascendente, en la Figura 13 se muestra el módulo de visualización de instancias.

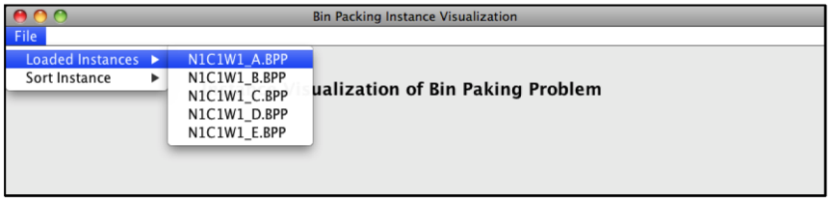


Figura 13 Módulo de Visualización de Instancias de VisTHAA [Castillo, 2011].

Una vez seleccionada la instancia a visualizar, la herramienta procede a visualizar la información de la instancia en una gráfica de barras como se muestra en la Figura 14.

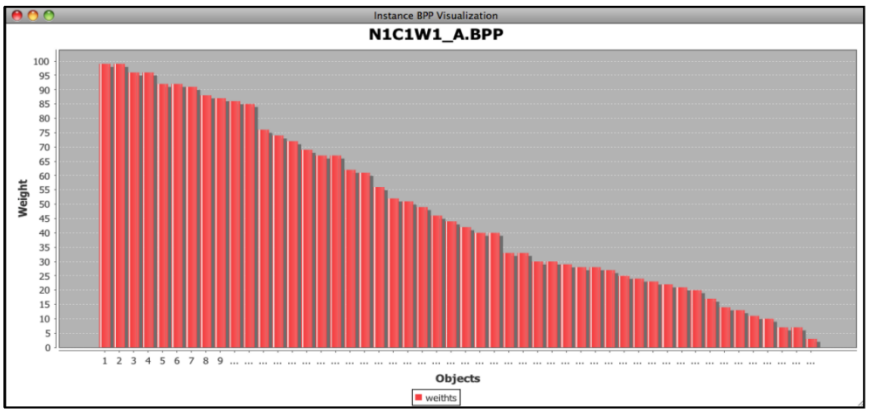


Figura 14 Visualización Gráfica de los datos de Instancias Cargadas [Castillo, 2011].

2.4.3.2 Instancias del Problema de Orden Ascendente

En la segunda opción mencionada en el punto anterior, la herramienta permite visualizar la información de las instancias cargadas, en la Figura 15 se muestra la selección de las instancias que se desea visualizar de forma ascendente.

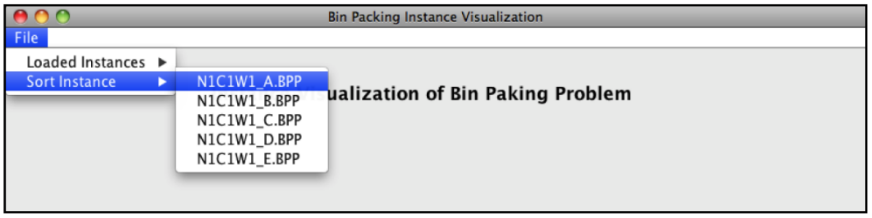


Figura 15 Módulo de Visualización de Instancias en forma Ascendente de VisTHAA [Castillo, 2011].

Una vez seleccionada la instancia a visualizar, la herramienta procede a visualizar la información de la instancia en una gráfica de barras en forma ascendente como se muestra en la Figura 16.

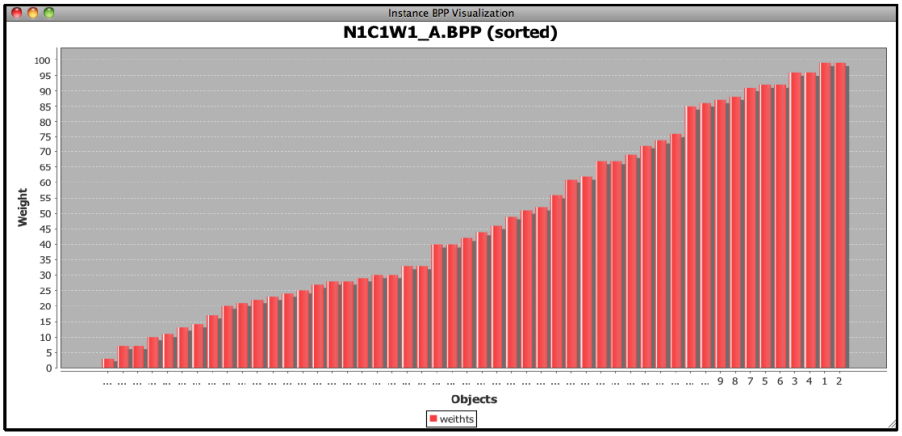


Figura 16 Visualización Gráfica de los Datos de Instancias Cargadas en forma Ascendente [Castillo, 2011].

2.4.3.3 Graficas de Frecuencia

Este tipo de gráficas permiten conocer la cantidad de objetos que se repiten dentro de un determinado porcentaje de la capacidad del contenedor y dentro de que rango se encuentran los objetos. En la Figura 17

se muestra la ruta del proceso de visualización de graficas de frecuencias de VisTHAA. El eje de las abscisas representa el peso de los objetos como porcentaje de la capacidad del contenedor, es decir $0 < \frac{w_i}{c} \leq 1$.

El eje vertical cuantifica el número de objetos que se localizan dentro de un intervalo de porcentaje [Quiroz 09]. El porcentaje de incremento es del 1%, lo que significa que se contabilizará cuantos objetos están entre el 0% y el 1%, cuantos objetos están entre el 1% y el 2% y así sucesivamente hasta llegar al 100%.

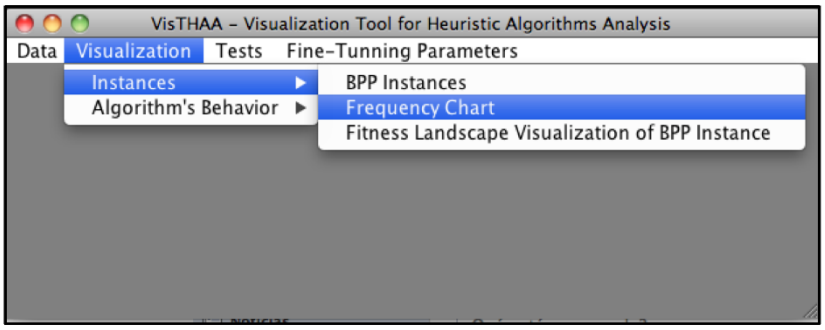


Figura 17 Ruta del Módulo de Visualización de Gráficas de Frecuencia [Castillo, 2011].

Una vez accedido al módulo lo que continúa es seleccionar la instancia específica a ser visualizada, la cual se muestra en la Figura 18.

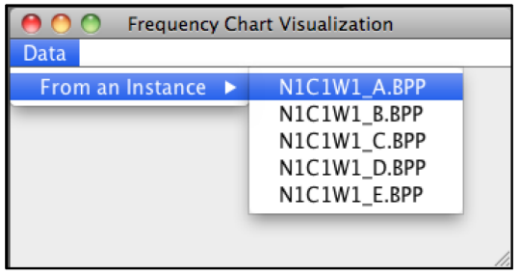


Figura 18 Módulo de Visualización de Gráficas de Frecuencia VisTHAA [Castillo, 2011].

Posterior a la selección de la instancia, se muestra la gráfica de frecuencias referente a ella, la cual se muestra en la Figura 19.

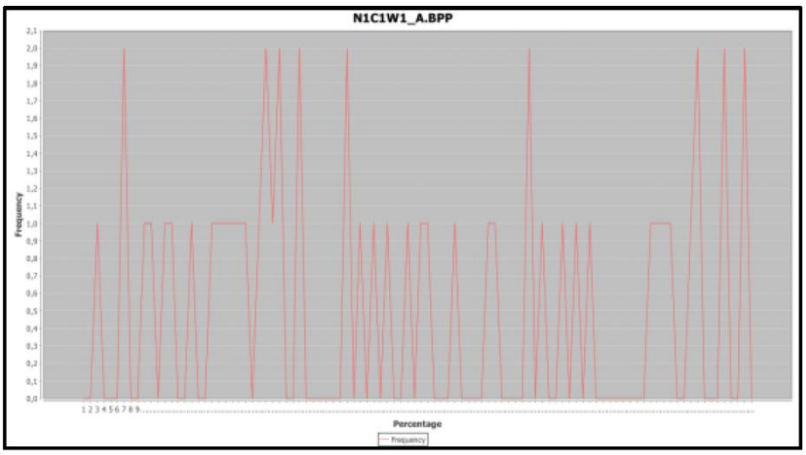


Figura 19 Visualización Gráfica del Peso de los Elementos de la Instancia N1C1W1_A.BPP [Castillo, 2011].

2.4.3.4 Graficas de Superficie de Aptitudes

En la tercera opción mencionada en el punto anterior, la herramienta permite visualizar la superficie de aptitudes a través de una caminata aleatoria. En la Figura 20 se muestra la ruta del módulo de visualización de superficie de aptitudes.

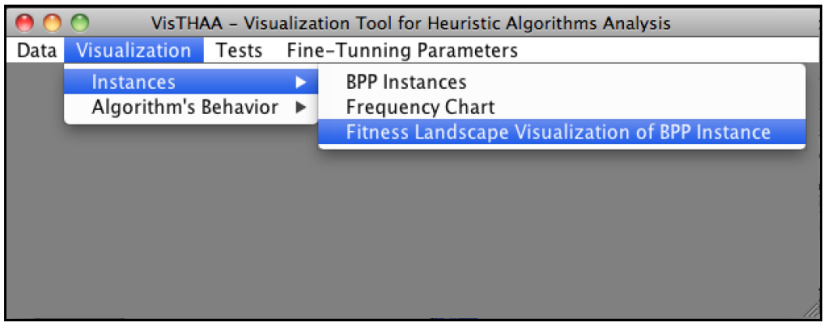


Figura 20 Ruta del Módulo de Visualización de la Superficie de Aptitudes [Castillo, 2011].

La visualización de la superficie de aptitudes se llevó al siguiente nivel, visualizarla en las tres dimensiones espaciales. La idea general del método es extraer una muestra representativa de la instancia a través de un algoritmo Random Walk. Luego, ese conjunto de datos unidimensional convertirlo en un conjunto bidimensional sin alterar de ninguna manera los valores de aptitud calculados. Finalmente, ya con tres variables por cada solución (valor de fila, valor de columna y valor de aptitud) es posible visualizar dichos puntos en el espacio.

Descripción general del Algoritmo Random Walk

La idea general del algoritmo *Random Walk* es obtener una solución inicial factible, y a partir de ella generar soluciones vecinas iterativamente hasta un cierto número de pasos predefinidos.

En cada paso de la caminata aleatoria una nueva solución es evaluada, sin embargo, no es de interés que el algoritmo reporte la mejor solución alcanzada durante su ejecución, dicho algoritmo fue implementado en el trabajo de [Castillo, 2011] el cual se muestra a continuación.

1. **Generar** una solución inicial factible.
2. **Computar** los valores de la función objetivo y la función de aptitud.
3. **Para** i = 1 hasta N - 1, hacer:
    Generar una solución vecina de la solución actual.
    Computar los valores de la función objetivo y la función de aptitud.
4. **Almacenar** los resultados en un archivo de texto plano.

Donde:

N es el número de pasos de la caminata aleatoria.

En la Figura 21 se observa en la parte izquierda de la ventana que aparecen las instancias que han sido cargadas a la herramienta, en la parte central algunos campos deben de ser configurados para la ejecución del algoritmo, una vez completado todos los campos se procede a hacer clic en el botón de "Execute Random Walk".

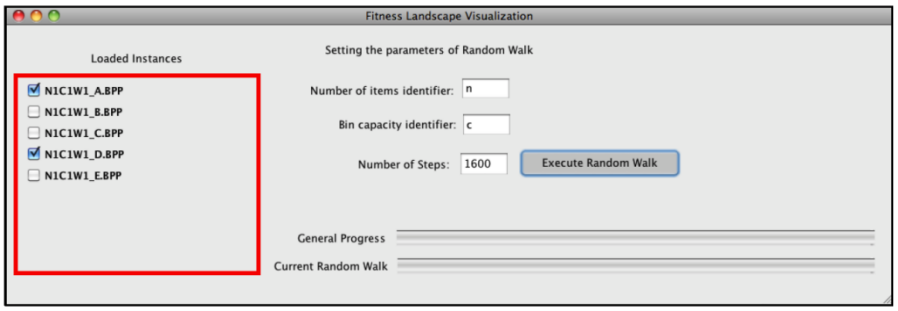


Figura 21 Módulo de Visualización de la Superficie de Aptitudes [Castillo, 2011].

En la Figura 22 se muestra una gráfica en 3D de la caminata aleatoria aplicado a una instancia del problema de empacado de contenedores.

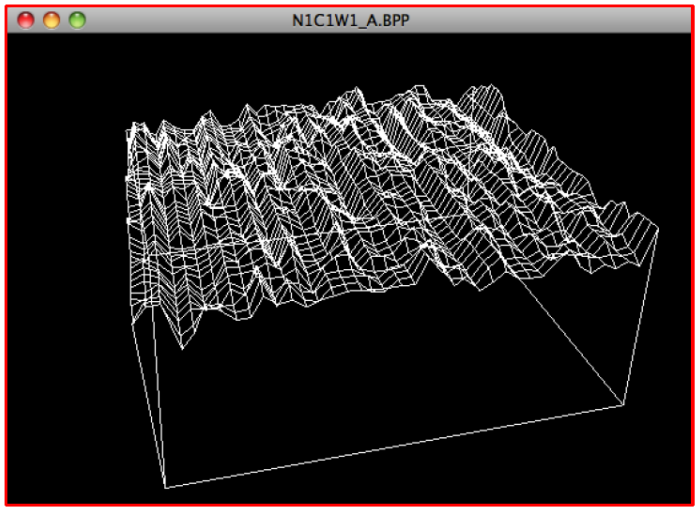


Figura 22 Visualización de la Superficie para la Instancia N1C1W1_A.BPP [Castillo, 2011].

2.4.4 Módulo de Análisis Estadístico

La función de este módulo permite la evaluación de desempeño de dos algoritmos, dicha evaluación es efectuada a través de la prueba de Wilcoxon, el cual es clasificado como un procedimiento no paramétrico que determina si dos conjuntos de datos muestran una diferencia significativa [Mendenhall, 1997].

La prueba de Wilcoxon se utiliza para determinar si dos muestras representan dos diferentes poblaciones [García 08]. Específicamente en el ámbito de las ciencias computacionales, la prueba de Wilcoxon se utiliza para determinar si las diferencias en el desempeño de dos algoritmos son estadísticamente significativas, ya que, de ser así, se puede afirmar fehacientemente, con un nivel de confianza determinado, que un algoritmo es superior a otro.

Debido a lo anterior, se implementó un módulo en la herramienta VisTHAA que permitiera determinar si un algoritmo es superior a otro estadísticamente, a través de la aplicación de la prueba de Wilcoxon. En la Figura 23 se muestra la ruta del módulo de prueba estadística de Wilcoxon.

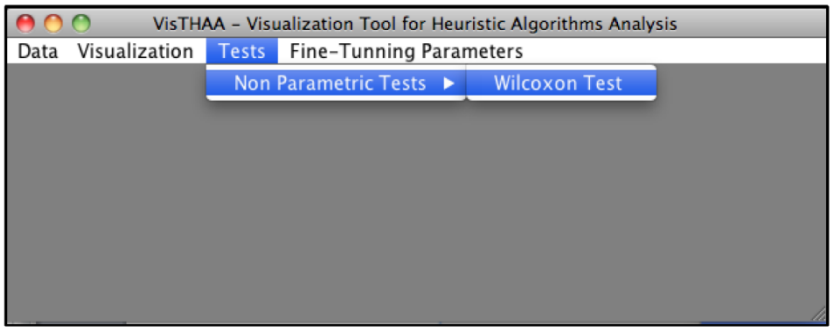


Figura 23 Ruta del Módulo de Prueba de Wilcoxon [Castillo, 2011].

Una vez accedido al módulo, se muestra una ventana cuyo principal objetivo es cargar a VisTHAA los datos del desempeño de los algoritmos, para esto accedemos al menú *Load,* y luego al menú *...from a File*, tal y como se muestra en la Figura 24.

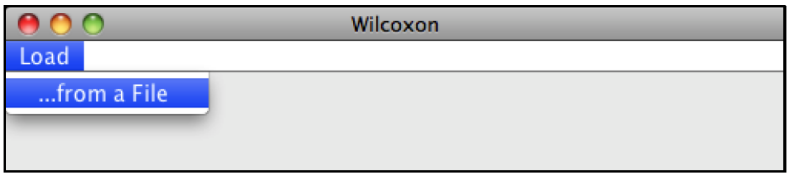


Figura 24 Módulo de Prueba Paramétrica de Wilcoxon [Castillo, 2011].

En la Figura 25 se muestra un navegador de archivos, dentro del cual, al igual que las instancias, debemos buscar el archivo correspondiente, en este caso, el archivo donde se encuentren los datos de desempeño de los algoritmos a ser evaluados.

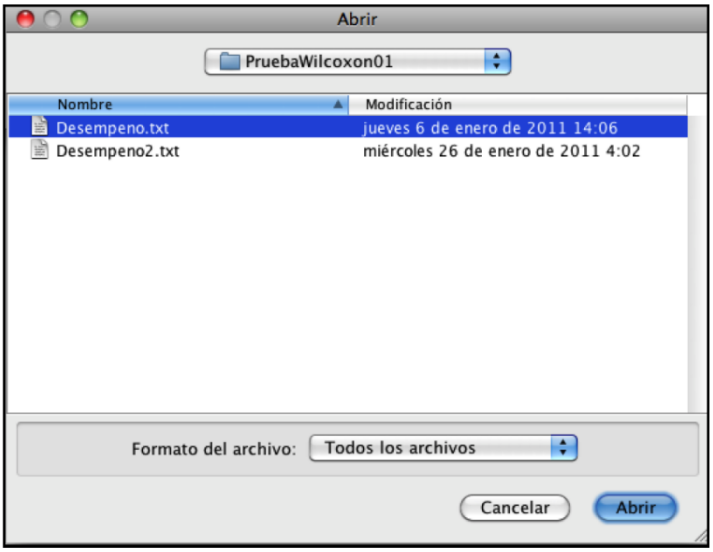


Figura 25 Ventana de Búsqueda de Archivo de Desempeño [Castillo, 2011].

Cabe mencionar que los datos de desempeño se toman de un archivo de texto plano el cual únicamente debe tener dos líneas, donde la primera línea debe contener un conjunto de datos del desempeño del primer algoritmo, mientras que en la segunda línea debe contener los datos del desempeño del segundo algoritmo.

Una vez hecho esto, una nueva ventana es mostrada, ya con los datos del archivo seleccionado, ahora únicamente hace falta hacer clic en el botón *Execute Wilcoxon Test* para que VisTHAA automáticamente realice dicha prueba. La Figura 26 muestra la ventana principal de la prueba de Wilcoxon.

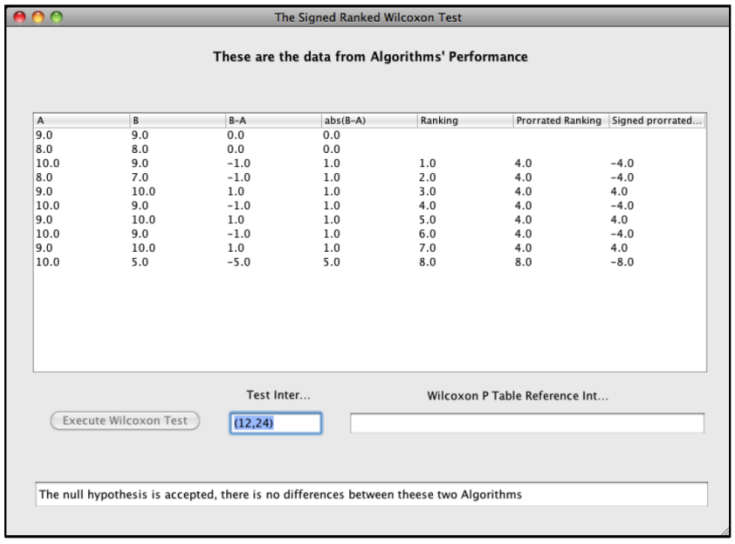


Figura 26 Ventana de Resultados de la Prueba de Wilcoxon [Castillo, 2011].

2.4.5 Módulo de rediseño causal

Permite la visualización de los resultados obtenidos en cada una de las fases anteriormente descritas, aplicadas al problema de selección de portafolio de proyectos multi objetivo y el algoritmo NSGA-II. En la

Figura 27 se muestra la interfaz del módulo de rediseño algorítmico incorporado en VisTHAA, el cual está compuesto por dos paneles, el panel izquierdo contiene el conjunto de instancias a analizar dependiendo al problema, y el panel derecho muestra el conjunto de algoritmo solucionadores.

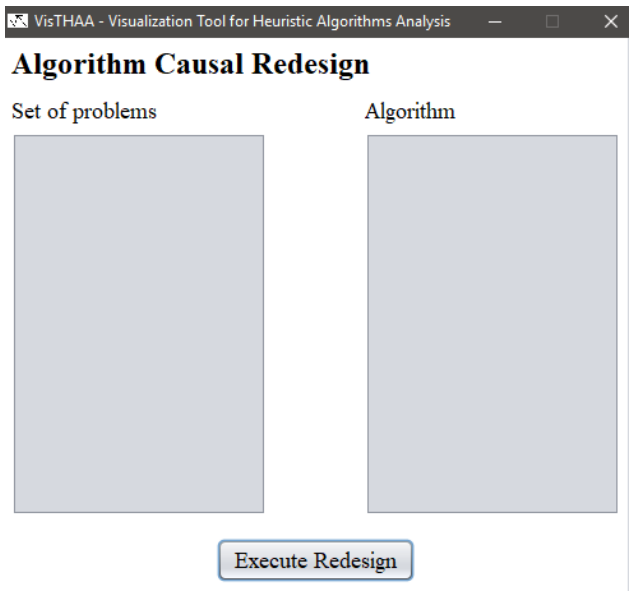


Figura 27 Ventana Algorithm Causal Redesign [García, 2019].

2.5 Framework

Es un marco o esquema de trabajo generalmente utilizado por programadores para realizar el desarrollo de software. Utilizar un framework permite agilizar los procesos de desarrollo ya que evita tener que escribir código de forma repetitiva, asegura unas buenas prácticas y la consistencia del código [Haro, 2008].

Un framework es por tanto un conjunto de herramientas y módulos que pueden ser reutilizados para varios proyectos, a continuación, se listan algunas ventajas de su uso.

El programador ahorra tiempo ya que dispone ya del esqueleto sobre el que desarrollar una aplicación.

- Facilita los desarrollos colaborativos, al dejar definidos unos estándares de programación.
- Al estar ampliamente extendido, es más fácil encontrar herramientas, módulos e información para utilizarlo.
- Proporciona mayor seguridad, al tener gran parte de las potenciales vulnerabilidades resueltas.
- Normalmente existe una comunidad detrás, un conjunto de desarrolladores que pueden ayudar a responder consultas.

M-SDOSS

En la Figura 28 se presenta la metodología utilizada para seleccionar los patrones de diseño más aptos, en la construcción de la arquitectura del Framework.

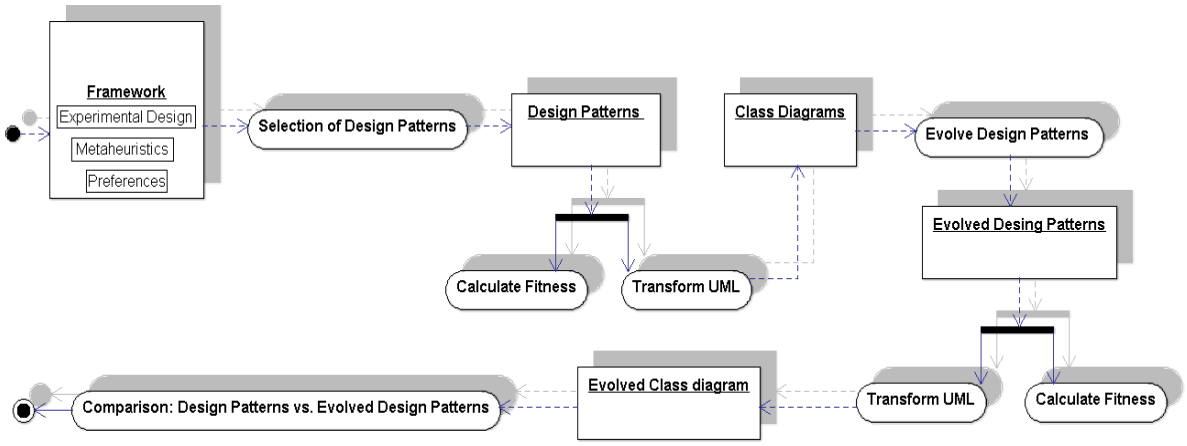


Figura 28 Metodología de patrones de diseño para el Framework de optimización [Rojas, 2017]

Los patrones de diseño seleccionados permitirán dar mayor flexibilidad en la creación y modificación de las clases pertenecientes a los módulos de optimización. La metodología propuesta se describe brevemente en la Tabla 3.

Tabla 3 Descripción de la metodología

Proceso Evolutivo	
Etapa	**Descripción**
1	Se selecciona los patrones de diseño que permitan manipular los módulos de optimización del Framework, calculando posteriormente su *fitness* (beneficio).
2	Se procede a transformar los patrones de diseño a diagramas de clases UML.
3	Mediante un algoritmo genético, se procede a evolucionar los patrones de diseño elegidos en la etapa 1; calculando nuevamente su *fitness.*
4	Se procede transformar los patrones de diseño evolucionados a diagramas de clases UML. Dando como resultado un diagrama de clases evolucionado.
5	Por último, se compararán los patrones de diseño de la primera etapa contra los de la tercera etapa. Esto con el fin de observar el impacto que existe en utilizar unos u otros patrones de diseño en la arquitectura del Framework.

Enseguida se detallan las etapas utilizadas en la metodología descrita de la Tabla 4:

- **Etapa 1.-** Las Figuras 29 y 28 muestran algunos ejemplos de los patrones de diseño seleccionados, los cuales se marcan con un círculo azul las clases que hacen referencia a dichos patrones.

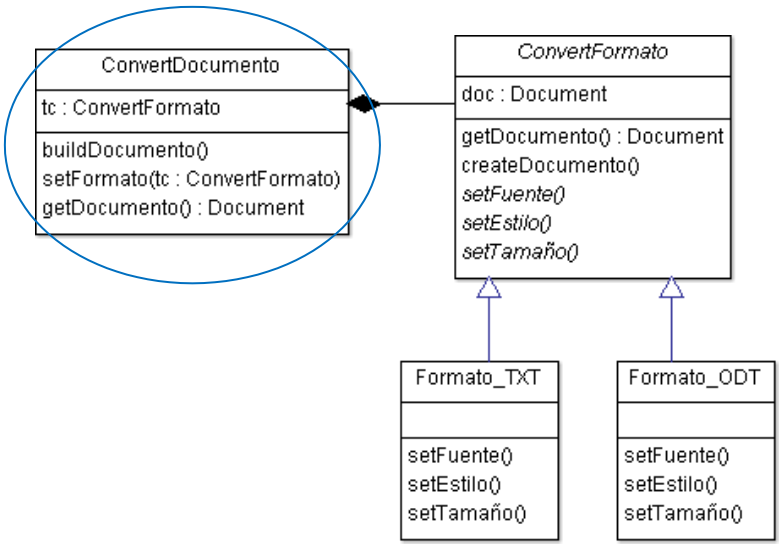


Figura 29 . Ejemplo del patrón de diseño Builder [Rojas, 2017]

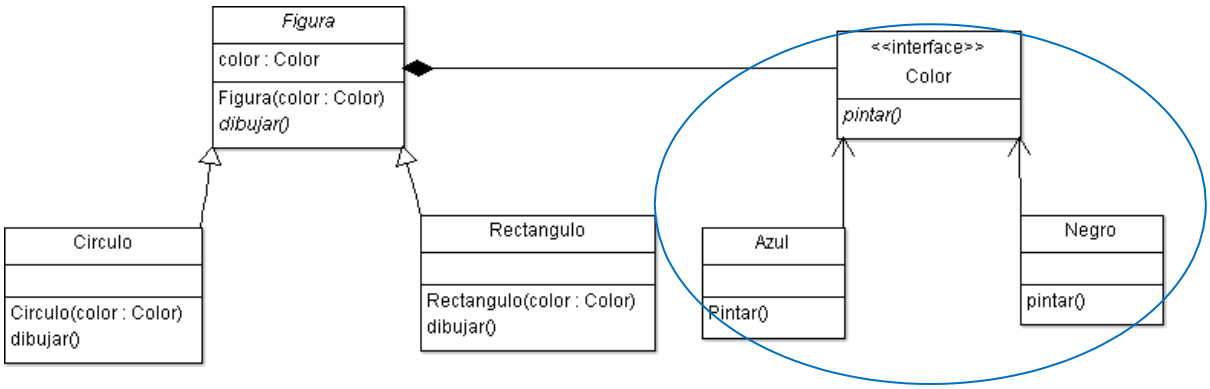


Figura 30 Ejemplo del patrón de diseño Bridge [Rojas, 2017]

- **Etapa 2.-** Se procede a transformar los patrones seleccionados a diagramas de clases UML. Las Figuras 31 y 32, muestran dos posibles maneras diseñar la arquitectura del Framework.

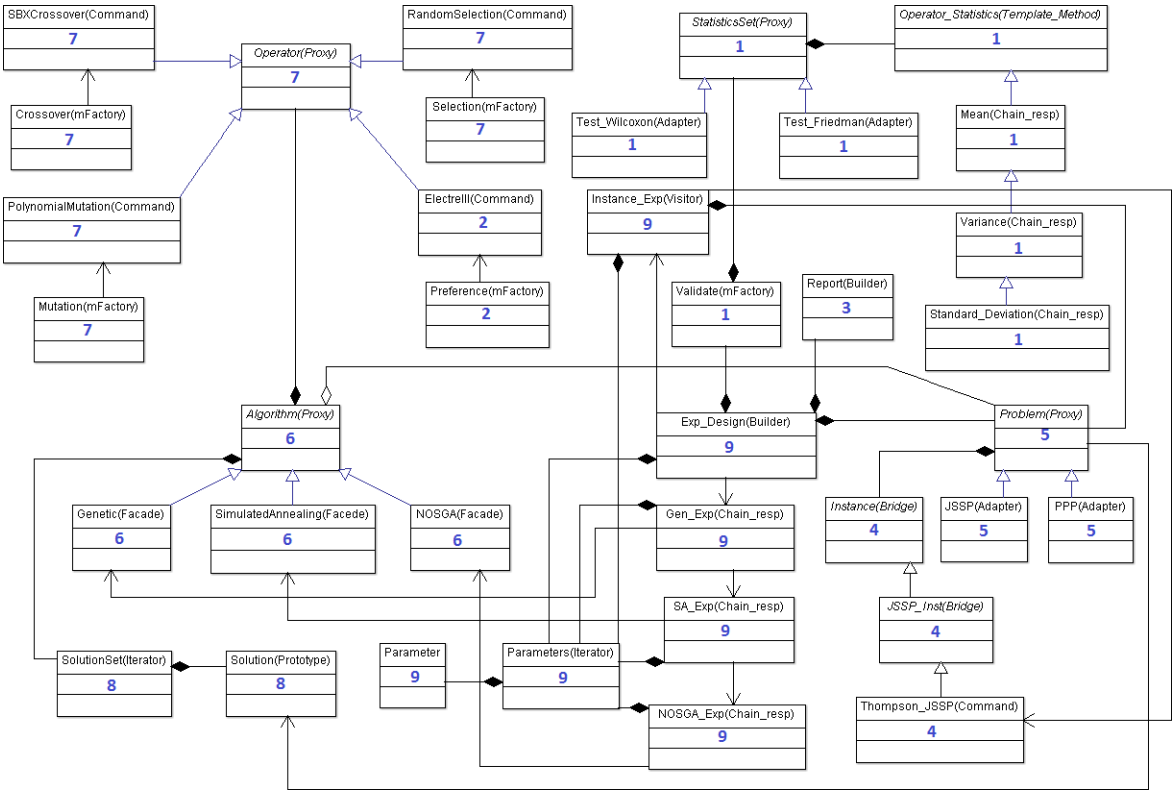


Figura 31 Primera arquitectura del Framework de optimización [Rojas, 2017].

Los patrones seleccionados fueron elegidos estratégicamente para maximizar la reutilización y extensibilidad del Framework. Por ejemplo, gracias a la implementación del patrón de diseño **proxy** en la clase *Problem*, los algoritmos podrán acceder a sus clases heredadas, sin que estas sean declaradas en los algoritmos. De la misma forma, las demás clases del Framework son diseñadas con patrones de diseño, pudiendo impactar en gran medida en algunos criterios de modularidad, como es la *cohesión* y *acoplamiento*.

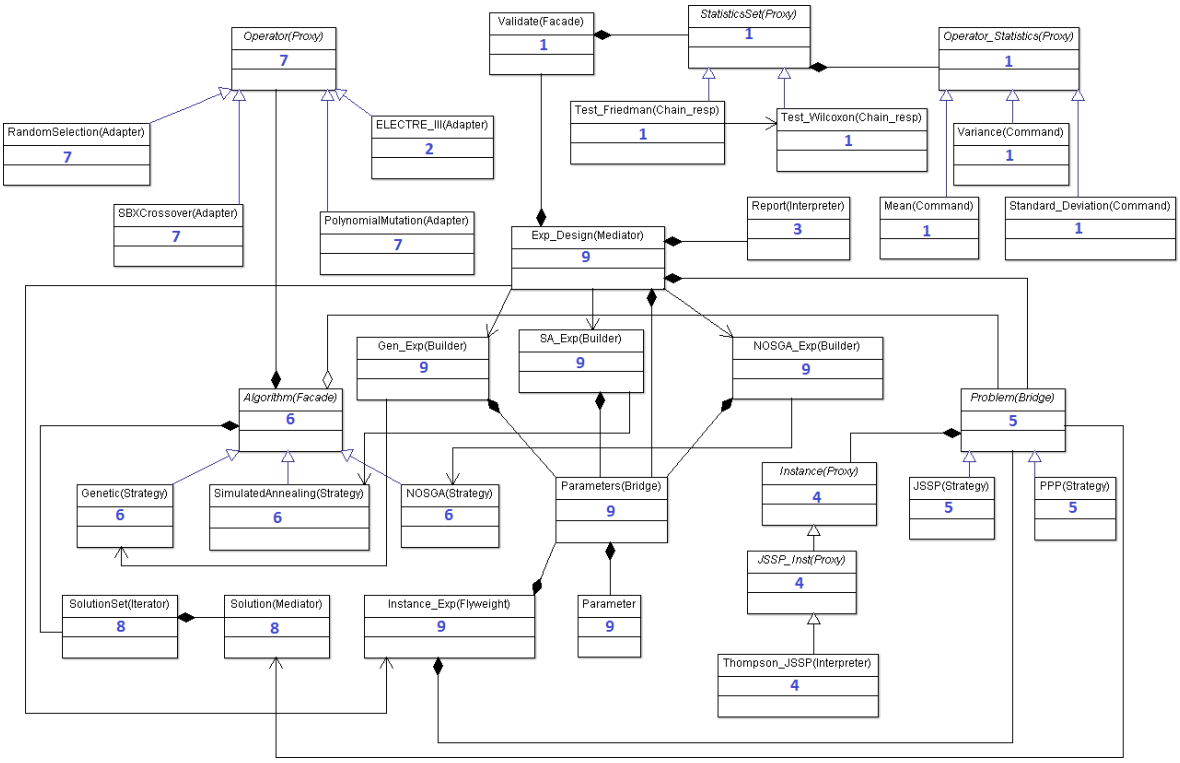


Figura 32 Segunda arquitectura del Framework de optimización [Rojas, 2017].

La Tabla 4 describe la funcionalidad que tiene cada uno de los módulos pertenecientes al Framework de optimización; nombrando las clases más sobresalientes de cada uno.

Tabla 4 Módulos de optimización del Framework

Características de los módulos de optimización	
Módulo	**Descripción**
Validación **1**	Mediante el uso de pruebas estadísticas como Friedman y Wilcoxon, permitirá seleccionar las mejores soluciones creadas por los algoritmos, en función de cada problema. Además, mediante la generalización algunos operadores estadísticos como la media, varianza y desviación estándar, facilitara la creación de nuevas pruebas estadísticas. Este módulo este compuesto por las clases: *Validate*, *StatisticsSet*, *Test_Friedman*, *Test_Wilcoxon*, *Operator_Statistics*, *Mean*, *Variance* y *Standard_Desviation*.
Preferencia **2**	Este módulo aborda un sistema relacional de preferencias compuesto de varias relaciones binarias [Roy, 1996] como: Indiferencia, Preferencia estricta, Preferencia débil, Incomparabilidad, K-preferencia y No preferencia. Este módulo ayudará a implementar diferentes técnicas pertenecientes a MCDA; el cual se compone por las clases: *Preference* y *ELECTRE III*.

Reporte **3**	Este módulo presentara los resultados obtenidos del módulo "Validación", de tal manera que puedan ser revisados y comparados para su posterior análisis. Este módulo está compuesto por la clase: *Report*.
Instancia **4**	Este módulo está diseñado para contener la instancia de cualquier problema, de tal manera al generalizar su información, pueda ser abordado por el módulo "Problema"; sin importar el formato que utilicen los autores de cada instancia. Actualmente este módulo está compuesto por las clases: *Instance, JSSP_Inst* y *Thompson_JSSP*.
Problema **5**	Este módulo permite generalizar distintos problemas de optimización, enfatizando sus características comunes. Actualmente este módulo está compuesto por las clases: *Problem, JSSP* y *PPP*.
Algoritmo **6**	Este módulo permite manipular la información generada por módulo "Problema", según el comportamiento de cada algoritmo. Este permite generalizar distintos algoritmos, enfatizando sus características comunes, facilitando la creación y extensión de sus clases. Este módulo está compuesto por las clases: Algorithm, *Genetic*, *SimulatedAnnealing* y *NOSGA.*
Operadores **7**	Este módulo contiene un conjunto de operadores en el área de optimización, como p. e. Cruza, Mutación, Selección, etc. Estos son necesarios para llevar a cabo la ejecución de múltiples algoritmos como p.e. el Genético, Recosido Simulado y NOSGA (todos estos dentro del módulo "Algoritmo"). Este módulo está compuesto por las clases: *Operator*, *Crossover*, *Mutation*, *Selection*, etc.
Solución **8**	Este módulo permite generalizar las soluciones creadas por el módulo "Algoritmo"; que, de otra manera, se tendría que crear tantas clases de soluciones como existan algoritmos en el Framework. Este módulo está compuesto por las clases: *Solution* y *SolutionSet*.
Diseño Experimental **9**	Este módulo es uno de los más importantes de este Framework, ya que permite realizar una experimentación sencilla y cómoda. Además, también permite configurar fácilmente los parámetros pertenecientes a módulos de optimización (p. e. "Problema", "Algoritmo", etc.). De esta manera evitemos la configuración directa a esos módulos. Actualmente este módulo está compuesto por las clases: *Parameter*, *Parameters*, *Exp_Design*, *Gen_Exp*, *SA_Exp* y *NOSGA_Exp* e *Instance_Exp*.

En la Tabla 5 se presenta dos posibles maneras de diseñar la arquitectura del Framework, mediante el uso de patrones de diseño.

Tabla 5 Caracterización del Framework de optimización.

Framework					
Module		**Class**		**Design patterns**	
ID	**Name**	**Package**	**Name**	**Option 1**	**Option 2**
1	**Validación**	1	*Operator_Statistic*	Template method	Proxy
			Mean	Chain responsibility	Command
			Variance	Chain responsibility	Command
			Standard_Desviation	Chain responsibility	Command
		2	Validate	Factory method	Facade
			StatisticSet	Proxy	Proxy
			Test_Fridman	Adapter	Chain responsibility
			Test_Wilcoxon	Adapter	Chain responsibility
2	**Preferencia**	3	Preference	Factory method	
			ELECTRE III	Command	Adapter
3	**Reporte**	4	Report	Builder	Interpreter
4	**Instancia**	5	*Instance*	Bridge	Proxy
			JSSP_Inst	Bridge	Proxy
			Thompson_JSSP	Command	Interpreter
5	**Problema**	6	*Problem*	Proxy	Bridge
			JSSP	Adapter	Strategy
			PPP	Adapter	Strategy
6	**Algoritmo**	7	*Algorithm*	Proxy	Facade
			Genetic	Facade	Strategy
			SimulatedAnnealing	Facade	Strategy
			Nosga	Facade	Strategy
7	**Operadores**	8	*Operator*	Proxy	Proxy
			Crossover	Factory method	Adapter
			Mutation	Factory method	Adapter

			Selection	Factory method	Adapter
			SBXCrossover	Command	
			PolynomialMutation	Command	
			RandomSelection	Command	
8	**Solución**	9	Solution	Prototype	Madiator
			SolutionSet	Iterator	Iterator
9	**Diseño Experimental**	10	Exp_Desing	Builder	Madiator
			Gen_Exp	Chain responsibility	Builder
			SA_Exp	Chain responsibility	Builder
			NOSGA_Exp	Chain responsibility	Builder
			Instance_Exp	Visitor	Flyweight
		11	Parameters	Iterator	Bridge
			Parameter		

2.6 Problemas de optimización

Un problema de optimización consiste en encontrar la mejor solución posible dentro de un espacio de soluciones, en donde la mejor solución es considerada con respecto a un criterio máximo o mínimo. En este trabajo de investigación se abordarán los siguientes casos de estudio:

1. El problema de la mochila mono objetivo (Knapsack problem).
2. El problema de selección de cartera de proyectos multi objetivo.
3. La familia de problemas DTLZ.

2.6.1 Problema de la mochila

El problema de la mochila es uno de los problemas clásicos de optimización combinatoria, ya que es clasificado como un problema NP-Duro [Cruz, 2004]. Es un problema de optimización combinatoria que juega un rol importante en la teoría de la computación, ya que tiene como meta la búsqueda de la mejor solución entre un conjunto finito de posibles soluciones a un problema.

El objetivo del problema es seleccionar la mayor cantidad de objetos, que se llevarán en la mochila, maximizando su beneficio sin sobrepasar la capacidad de [Martello, 1990].

$$maximizarZ = \sum_{j=1}^{N} C_j X_j \tag{2}$$

$$s.a \sum_{j=1}^{N} x_j p_j \leq P \tag{3}$$

$$X_j \in \{0,1\}, j \in N = \{1,2,\dots,n\} \tag{4}$$

Donde:

- La función objetivo (2) maximiza el beneficio total Z del beneficio de los objetos C_j asignados a la mochila.
- La restricción (3) asegura que el peso total p_j de los objetos x_j no sobrepase la capacidad de *P.*
- La restricción (4) exige que las variables de decisión sean binarias, si x_j es tal que $x_j = 0$ si el objeto *j* no está en la mochila, $x_j = 1$ en caso contrario.

2.6.2 Problema de selección de carteras de proyectos

La correcta selección de proyectos para integrar una cartera de proyectos es uno de los problemas más importantes de decisión, tanto para instituciones públicas como privadas.

Los principales modelos económicos y matemáticos para el problema de cartera de proyectos suponen que se tiene un conjunto definido *N* de proyectos, cada proyecto perfectamente caracterizado con costos e ingresos, de los cuales la distribución en el tiempo es conocido.

En caso de riesgos, el "Tomador de Decisiones" (DM) debe conocer la probabilidad de distribución de los beneficios. El DM es una persona o un grupo de personas a cargo de seleccionar, a su juicio y dada su experiencia, las mejores soluciones. [Fernández, 2001]. El modelo matemático del problema se expresa de la siguiente manera:

$$maximizarZ = \sum_{j=1}^{N} c_j x_j \tag{5}$$

Donde:

- La función objetivo (5) maximiza el beneficio total asociado Z a una cierta cartera de proyectos, c_j es el beneficio asociado con el proyecto j y $x_j = 1$ indica si el proyecto j forma parte de la cartera y $x_j = 0$ en caso contrario.
- Sea x(j) la función indicadora del conjunto de los proyectos finalizados donde $x(j) = 1$ si el proyecto j recibe el financiamiento solicitado y $x(j) = 0$ en caso contrario.

2.6.3 Problemas DTLZ

El banco de funciones DTLZ fue propuesto originalmente por [Deb, 2005] se incluye las siguientes funciones continuas.

2.6.3.1 Problema de prueba DTLZ1

Como problema de prueba simple, se construye un problema de objetivo M con un frente de Pareto optimo lineal:

$$\left.\begin{array}{ll} min & f_1(X) = \frac{1}{2}x_1x_2 \dots x_{M-1}\left(1 + g(X_M)\right), \\ min & f_2(X) = \frac{1}{2}x_1x_2 \dots (1 - x_{M-1})\left(1 + g(X_M)\right), \\ \vdots & \vdots \\ min & f_{M-1}(X) = \frac{1}{2}x_1(1 - x_2)\left(1 + g(X_M)\right), \\ min & f_M(X) = \frac{1}{2}(1 - x_1)\left(1 + g(x_M)\right), \\ subject & 0 \le x_i \le 1, fori = 1,2, \dots, n. \end{array}\right\} \quad (6)$$

La función $g(X_M)$ requiere $|X_M| = k$ variables y debe tomar cualquier función $g \geq 0$.Se sugiere lo siguiente como se observa en la ecuación 6.

$$g(X_M) = 100\left[|X_M| + \sum_{x_i \in X_M}(x_i - 0.5)^2 - cos\big(20\pi(x_i - 0.5)\big)\right] \quad (7)$$

2.6.3.2 Problema de prueba DTLZ2

Este problema es idéntico al problema genérico de la esfera, la construcción se muestra a continuación

$$\begin{array}{ll} min & f_1(X) = \left(1 + g(X_M)\right) cos\left(\frac{x_1\pi}{2}\right) cos\left(\frac{x_2\pi}{2}\right) \dots cos\left(\frac{x_{M-2}\pi}{2}\right) cos\left(\frac{x_{M-1}\pi}{2}\right), \\ min & f_2(X) = \left(1 + g(X_M)\right) cos\left(\frac{x_1\pi}{2}\right) cos\left(\frac{x_2\pi}{2}\right) \dots cos\left(\frac{x_{M-2}\pi}{2}\right) sin\left(\frac{x_{M-1}\pi}{2}\right), \\ min & f_3(X) = \left(1 + g(X_M)\right) cos\left(\frac{x_1\pi}{2}\right) cos\left(\frac{x_2\pi}{2}\right) \dots s \in \left(\frac{x_{M-2}\pi}{2}\right), \\ \vdots & \vdots \\ min & f_{M-1}(X) = \left(1 + g(X_M)\right) cos\left(\frac{x_1\pi}{2}\right) s \in \left(\frac{x_2\pi}{2}\right), \\ min & f_M(X) = \left(1 + g(X_M)\right) s \in \left(\frac{x_1\pi}{2}\right), \\ subject & 0 \le x_i \le 1, fori = 1,2, \dots, n, \\ & g(X_M) = \sum_{x_i \in X_M}\left(x_i - 0.5\right)^2. \end{array} \quad (8)$$

(9)

$$\sum_{i=1}^{M}(f_i\ )^2 = 1.$$

Las soluciones óptimas de Pareto corresponden a $x_M = 0.5$ y todos los valores de las funciones objetivos deben satisfacer la ecuación 9. Se recomienda que $k = |X_M| = 10$. El total de variables es $n = M + k - 1$.

2.6.3.3 Problema de prueba DTLZ3

Este problema fue diseñado principalmente para probar la habilidad de los MOEA's para converger hacia el frente de Pareto global óptimo:

$$\begin{aligned}
min \quad & f_1(X) = \left(1 + g(X_M)\right) cos\left(\frac{x_1\pi}{2}\right) cos\left(\frac{x_2\pi}{2}\right) \dots cos\left(\frac{x_{M-2}\pi}{2}\right) cos\left(\frac{x_{M-1}\pi}{2}\right),\\
min \quad & f_2(X) = \left(1 + g(X_M)\right) cos\left(\frac{x_1\pi}{2}\right) cos\left(\frac{x_2\pi}{2}\right) \dots cos\left(\frac{x_{M-2}\pi}{2}\right) sin\left(\frac{x_{M-1}\pi}{2}\right),\\
min \quad & f_3(X) = \left(1 + g(X_M)\right) cos\left(\frac{x_1\pi}{2}\right) cos\left(\frac{x_2\pi}{2}\right) \dots s \in \left(\frac{x_{M-2}\pi}{2}\right),\\
\vdots \quad & \vdots\\
min \quad & f_{M-1}(X) = \left(1 + g(X_M)\right) cos\left(\frac{x_1\pi}{2}\right) s \in \left(\frac{x_2\pi}{2}\right),\\
min \quad & f_M(X) = \left(1 + g(X_M)\right) s \in \left(\frac{x_1\pi}{2}\right),\\
& 0 \le x_i \le 1, for i = 1,2, \dots, n,\\
subject \quad & g(X_M) = 100\left[|X_M| + \sum_{x_i \in X_M} \left(x_i - 0.5\right)^2 - cos\left(20\pi\left(x_i - 0.5\right)\right)\right]
\end{aligned} \tag{10}$$

Se sugiere que $k = |X_M| = 10$. Existe un total de $n = M + k - 1$ variables de decisión en este problema. La función g anterior introduce $(3k - 1)$ frentes óptimos de Pareto locales y uno frente óptimo de Pareto global. Todos los frentes óptimos de Pareto locales son paralelos al óptimo Frente de Pareto global y un MOEA pueden atascarse en cualquiera de estos frentes óptimos de Pareto locales, antes de converger al frente óptimo de Pareto global en $(g * 0)$.

2.7 Metaheurísticas de solución

En esta sección se presentan los algoritmos solucionadores utilizados en el caso de estudio, siendo el algoritmo genético usado para solucionar el problema de la mochila mono objetivo y el NSGA-II para solucionar tanto el PSP como los DTLZ 1, 2 Y 3.

2.7.1 Algoritmos genéticos

Los algoritmos genéticos (Genetic Algorithm - GA) fueron presentados por J. Holland [Holland, 1995]. Estos tipos de métodos se basan en la recombinación de soluciones candidatas de una población, que evoluciona por medio de mecanismos genéticos como lo es la selección, la cruza y la mutación de los individuos de una población.

El concepto de recombinación de soluciones supone una de las aportaciones fundamentales de GA. Por otro lado, es también relevante la diferencia explícita entre la representación del problema (denominada genotipo), que habitualmente viene determinada por cadenas de bits conocidas como cromosomas, y las variables del problema en sí (denominadas fenotipo). GA representa una metaheurística poblacional sencilla e intuitiva que, muy probablemente, sea la más utilizada. El principio de operación es el siguiente:

Algoritmo 1. Algoritmo genético [Rivero, 2010]

1. **INICIO**
2. **INICIALIZAR** población actual aleatoriamente
3. **MIENTRAS** no se cumpla el criterio de terminación
4. **CREAR** población temporal vacía
5. **SI** elitismo: copiar en población temporal mejores individuos
6. **MIENTRAS** población temporal no llena
7. **SELECCIONAR** padres cruzar padres con probabilidad Pc
8. **SI** se ha producido el cruce
    i. **MUTAR** uno de los descendientes (prob. Pm)
    ii. **EVALUAR** descendientes
    iii. **AÑADIR** descendientes a la población temporal
9. **SINO**
    i. **AÑADIR** padres a la población temporal
10. **FIN SI**
11. **FIN MIENTRAS**
12. **AUMENTAR** contador generaciones
13. **ESTABLECER** como nueva población actual la población temporal
14. **FIN MIENTRAS**
15. **FIN**

2.7.2 Algoritmo genético de ordenamiento por no dominancia

Es un algoritmo genético popular basado en la no dominancia para la optimización multi objetivo [Srinivas, 1994], el cual tiene incorporado un algoritmo de ordenamiento y elitismo [Deb, 2002]. En el Algoritmo 2 se muestra el proceso de solución del NSGA-II.

El algoritmo NSGA-II, crea una población de padres de tamaño N (línea 0), la población se ordena de acuerdo con la dominancia de cada individuo (línea 3). Cada individuo se le asigna un valor de la aptitud en base a la dominancia. La población de hijos (Q_0) se genera mediante operadores genéticos de cruza y mutación. El procedimiento elitista para una generación $t \geq 1$ se realiza de la siguiente manera: se combina la población de padres e hijos, obteniendo una población de tamaño $2N$. La nueva población se ordena de acuerdo con la dominancia en diversos frentes [Deb, 2002]. La nueva población se forma con los individuos que forman parte del frente 0, frente 1 y así sucesivamente, hasta que el número de frentes sea igual o mayor a N [Balderas, 2018]. Las soluciones que cuentan con igual valor de dominancia formaran parte de un mismo frente (línea 3). La nueva población es formada seleccionando los individuos que pertenezcan a $F1$, y así sucesivamente, hasta que P_{T+1} sea igual o mayor a N (línea 5); los individuos restantes son rechazados.

Algoritmo 2. NSGA-II [Balderas, 2018]

Entrada: Iter*MAX*, *N*

Salida: *F0* de la última iteración del algoritmo

1. **Inicializar**: $P \leftarrow$ **PoblaciónAleatoria**(N), $Q \leftarrow \emptyset$
2. F**or** T=1 **to** Iter*MAX* **do**
3. RT $\leftarrow$ PT $\cup$ QT
4. $F \leftarrow$ **fast-non-dominated-sort** (RT)
5. i $\leftarrow$ 0, P$T+1 \leftarrow \emptyset$
6. **Mientras** $|PT+1|+|Fi| \leq N$
7. **crowding-distance-assignment** (Fi)
8. $PT+1 \leftarrow PT+1 \cup Fi$
9. i $\leftarrow$ i+1
10. F**in mientras**
11. **crowding-distance-sort** ($Fi, \prec n$) //ordenamiento ascendente
12. $PT+1 \leftarrow PT+1 \cup Fi$ [1: N – $|PT+1|$]
13. $QT+1 \leftarrow$ **generar-nueva-población** ($PT+1$)
14. **fin for**
15. **regresa** *F0*

2.8 Índices de caracterización

En esta sección se describirán algunas mediciones documentas en la literatura, que permiten caracterizar el desempeño de algoritmos en su versión mono objetivo y multi objetivo que son utilizados como parte de las experimentaciones como caso de estudio.

En la entrada, los casos proporcionan información acerca de la estructura de las instancias del problema; en el proceso los elementos principales son aquellos que proporcionan ideas acerca del comportamiento del algoritmo en ejecución; y en la salida, se buscan índices importantes del desempeño. Por lo que esto puede dar pie a analizando los diferentes indicadores se pueda realizar ajustes sobre la configuración de un experimento y mejorar sus resultados en algún caso particular.

2.8.1 Índices del desempeño de algoritmos mono objetivo

En esta sección se explican algunas mediciones propuestas en el trabajo de [Cruz, 2004], [Álvarez, 2006] y [Quiroz, 2009], las cuales se muestran en la Tabla 6, para la caracterización del desempeño de algoritmos mono objetivo.

Tabla 6 Índices de la Caracterización del Desempeño de Algoritmos Mono Objetivo

Nombre	Descripción	Formulación
Max_{MejorF}	Calcula el valor máximo de aptitud de una instancia.	$Max_{MejorF} = max(M)$
Min_{MejorF}	Este índice permite obtener el valor mínimo de aptitud de una instancia.	$Min_{MejorF} = min(M)$
Radio teórico	Este índice permite obtener la razón teórica donde Z_{enc} es el mejor valor obtenido en el algoritmo y Z_{opt} es el mejor valor reportado en la literatura para dicha instancia.	$Radio_{teorico} = \frac{Z_{enc}}{Z_{opt}}$
Error obtenido	Este índice calcula el porcentaje de error del valor obtenido del algoritmo	$desv_{apt} = \frac{(Z_{opt} - Z_{enc})}{Z_{opt}}$
Promedio de la función aptitud de las soluciones factibles	Índice muestra el promedio del desempeño de las soluciones factibles, en donde $Z_{factible}$ es el desempeño de una solución factible y pob es el tamaño de la población	$m_{\text{¦}} \frac{\sum_{i=1}^{pob} Z_{factible}}{pob}$
Varianza de la función aptitud de las soluciones factibles	Es la varianza de la aptitud de las soluciones factibles del algoritmo.	$var = \frac{1}{pob} \sum_{i=1}^{pob}$

2.8.2 Índices del desempeño de algoritmos multi objetivo

En la literatura se han documentado varias mediciones que permite la caracterización del desempeño de algoritmos que dan solución a problemas multi objetivo, el trabajo de Martínez [Martínez, 2017] propone la evaluación del rendimiento de algoritmos que dan solución a problemas multi objetivo.

El trabajo de [Martínez, 2017] clasifica las mediciones de desempeño en 3 categorías: la minimización de la distancia del frente de Pareto producido por el algoritmo con respecto al frente verdadero (suponiendo que se conoce), maximizar la distribución de soluciones obtenidas de tal manera que sea posible tener una distribución de vectores tan uniforme como sea posible y maximizar la cantidad de elementos del conjunto de óptimos de Pareto generados [Zitzler, 2000].

En la Tabla 7 se muestra un enlistado de mediciones para la evaluación del desempeño de algoritmos multi objetivos el cual se encuentran en la literatura.

Tabla 7 Índices de la Caracterización del Desempeño de Algoritmos Multi Objetivo

Nombre	Función	Formulación
Distancia Generacional [Van, 1998]	Mide la distancia de las soluciones que están en el conjunto de soluciones no dominadas encontradas hasta ahora de las del conjunto óptimo de Pareto Donde n es el número de soluciones en el conjunto de soluciones no dominadas encontrado hasta el momento y d_i es la distancia euclidiana (medido en el espacio objetivo) entre cada una de estas soluciones y el miembro más cercano del conjunto de Pareto óptima. Está claro que un valor de $GD = 0$ indica que todos los elementos generados están en el conjunto óptimo de Pareto. Con el fin de obtener resultados fiables, los conjuntos de soluciones no dominadas se normalizan antes de calcular la medida de distancia.	$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n}$
Spread [Deb, 2002]	Es una métrica de diversidad que mide el grado de propagación logrado entre las soluciones obtenidas. Donde d_i es la Distancia Euclidiana entre *N* soluciones consecutivas, $\overline{d}$ es la media de estas distancias, y d_f y d_l son las distancias euclidianas a las soluciones extremas (delimitador) del frente de Pareto exacta en el espacio objetivo. Esta métrica toma un valor de cero para una distribución ideal, señalando una extensión perfecta de las soluciones en el frente de Pareto. Se aplica esta métrica después de una normalización de los valores de la función objetivo.	$\Delta = d_f + d_l + \frac{\sum_{i=1}^{N-1} d_i - \overline{d} \vee}{d_f + d_l + (N-1)\overline{d}}$

Hipervolumen [Zitzler, 1999]	Es un índice combinado de convergencia y diversidad que calcula el volumen (en el espacio objetivo) cubierto por los miembros de un conjunto no dominado de soluciones Q. Matemáticamente, para cada solución $i \in Q$, un hipercubo v_i se construye con un punto de referencia W y la solución i como las esquinas diagonales del hipercubo.	$HV = volume$

Capítulo 3 Propuesta de solución

En este capítulo se presenta la metodología propuesta para el cumplimiento de los objetivos de este trabajo de tesis, el cual se conforma por un conjunto de subprocesos que permitirán la configuración, ejecución y análisis de experimentos computacionales mediante la comunicación entre VisTHAA y M-SDOSS.

Para la aplicación de estas metodologías, se requiere la información referente a los objetos usados por el framework para definir sus instancias, problemas, algoritmos y operadores, además de poder almacenar la información de configuración del experimento, indicadores de desempeño mono y multi objetivo, para posteriormente procesar la información para estar acorde al formato JSON y permitir la comunicación.

Previamente la herramienta VisTHAA ya contaba con un módulo para la selección de experimento computacional, pero esta era limitada y no permitía modificar la configuración del algoritmo, p. e., tamaño de la población entre otras. Por otra parte, el módulo de comunicación entre VisTHAA y un framework de optimización era algo que no existía y pretende robustecer la cantidad de algoritmos solucionadores con los que VisTHAA puede trabajar.

Las aportaciones realizadas en este trabajo para la herramienta VisTHAA (y M-SDOSS) se describen a continuación:

1. Análisis, diseño e implementación del módulo de asistencia de configuración de experimentos computacionales.
2. Análisis, diseño e implementación del módulo de comunicación entre VisTHAA y M-SDOSS
3. Análisis y rediseño del módulo de caracterización del desempeño de algoritmos, para procesamiento de archivos JSON, peticiones de ejecución y procesamiento de resultados de M-SDOSS.

En la Figura 33 se muestra de manera general la nueva arquitectura de VisTHAA/M-SDOSS, resaltando con borde punteado los módulos propuestos en este trabajo de tesis. Dichos módulos se describen a lo largo de este capítulo.

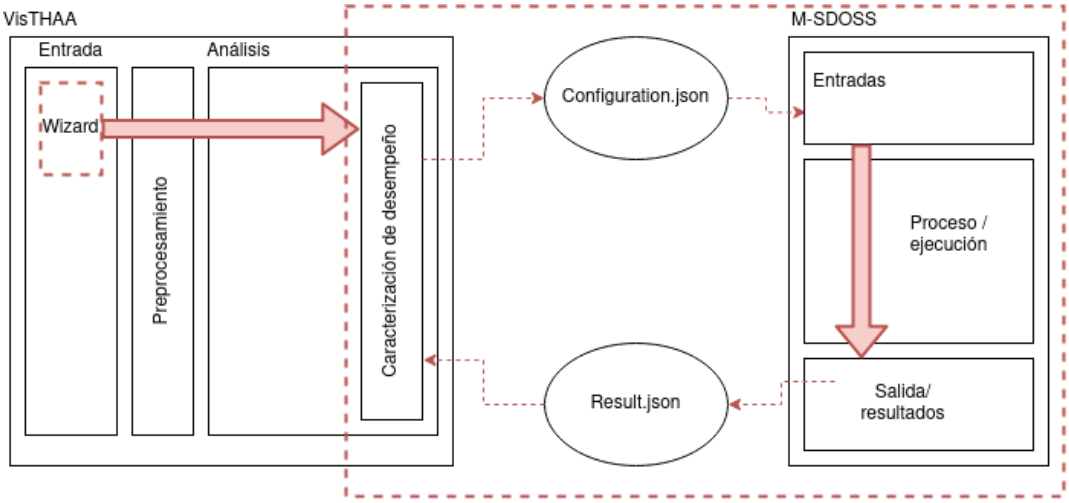


Figura 33 Diagrama general de VisTHAA/M-SDOSS

3.1 Metodología del módulo de configuración

En esta sección se da explicación de la metodología propuesta para el diseño del módulo de configuración de experimentos computacionales. En la Figura 34 se muestran cada una de las subsecciones para la definición de la configuración y primer paso del proceso de JSON de configuración.

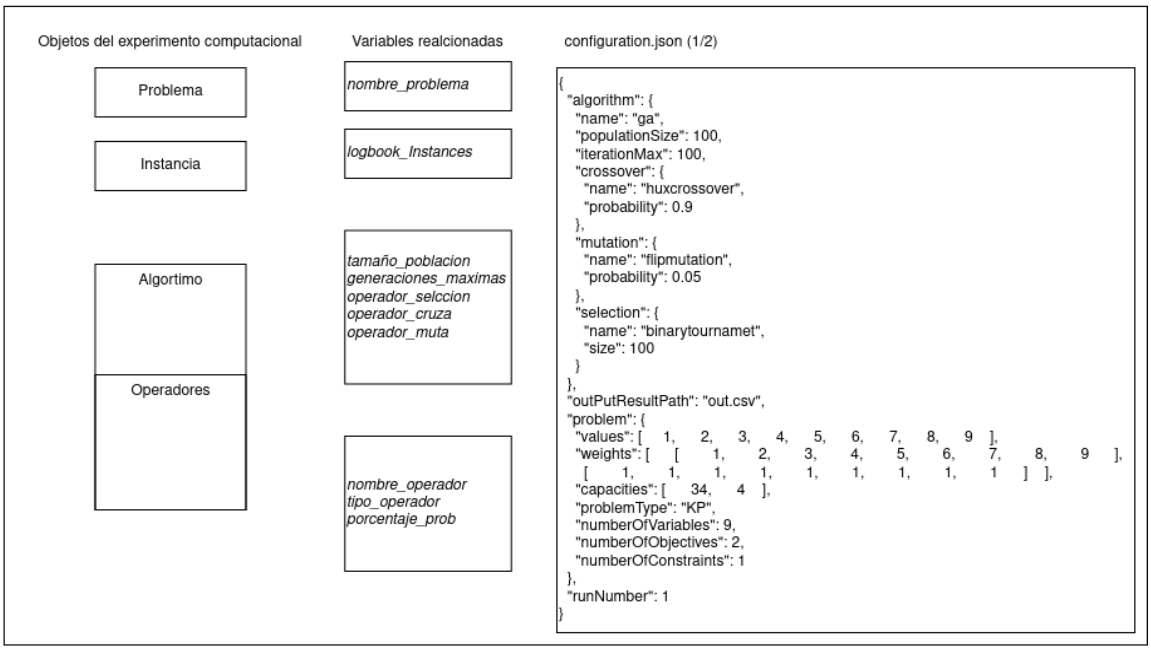


Figura 34 Objetos, variables y ejemplo del archivo JSON de configuración

Como se muestra en la Figura 34, los elementos que conforman el módulo de asistente en la configuración del experimento computacional de la herramienta VisTHAA, los cuales son el problema para resolver, el conjunto de instancias a procesar, la definición del algoritmo el cual contiene los atributos que se pueden configurar y el archivo JSON resultado del procesamiento

de esos valores, la definición de los objetos, así como la librería para manipulación de JSON son mostradas en el apéndice A.

En este trabajo se propone el diseño del asistente de configuración en base a la estructura del experimento computacional, la cual se encuentra resaltada en la línea punteada de la Figura 34 cada parte presentada como un objeto del experimento. En el algoritmo 3 se muestra el seguimiento del proceso de configuración del experimento encontrado en VisTHAA actualmente.

Algoritmo 3 Pasos del procesamiento de configuración

1. Inicio
2. Llamar al método GetProblemName()
3. Guardar el nombre del problema → problem
4. Llamar el método GetLogbook()
5. Guardar el nombre del archivo logbook →str
6. Llamar el método CompileLogbook(str)
7. Compila el archivo logbook
8. Llamar el método InstancesFunctions(str)
9. Guarda el número de instancias a preprocesar
10. Llamar el método InstructionsFileFunction(str)
11. Guarda el nombre del archivo metainstance
12. Llamar el método InstancesNamesFunction(str)
13. Guardar la ruta y el nombre del conjunto de instancias a preprocesar →instances
14. Llamar el método GetMetainstance()
15. Lee las variables del archivo metainstance
16. Llamar el método CalculateNumberOfVariables()
17. Guarda el número de variables→ n_variables
18. Llamar el método ResizeStructures(n_variables)
19. Redimensiona del conjunto de estructuras de almacenamiento del problema
20. Llamar el método LoadInstance (instances)
21. Guarda la información del conjunto de instancias a procesar en las estructuras de almacenamiento
22. Llamar el método ShowInstances(n_variables)
23. Imprime en pantalla las instancias cargadas
24. Llamar el método GetAlgorithm()
25. Guardar el algoritmo → Algorithm
26. Llamar el método GetPopSize()
27. Guardar el algoritmo → Algorithm
28. Llamar el método GetMaxIter()
29. Guardar el algoritmo → Algorithm
30. Llamar el método GetOperator(selection)
31. Guardar el operador → selection
32. Llamar el método GetOperator(mutation)
33. Guardar el operador → mutation
34. Llamar el método GetPercentage()
35. Convertir el porcentaje → realper
36. Llamar el método GetOperator(crossover)
37. Guardar el operador → crossover
38. Llamar el método GetPercentage()
39. Convertir el porcentaje → realper
40. Fin

A continuación, en la Tabla 8 se describen los métodos que conforman el módulo de asistente de configuración del experimento computacional que se encuentra actualmente en VisTHAA, incluyendo los que fueron reutilizados los cuales fueron desarrollados en el trabajo de [Castillo, 2011] y [García, 2019], y resaltando en negritas los que se diseñaron en este trabajo de tesis.

Tabla 8 Métodos del módulo de asistente de configuración de experimento computacional

Nombre del método	Descripción
GetProblem()	Obtiene el nombre del problema y prepara el objeto experiment
GetLogbook()	Lee el archivo "logbook.txt" y hace un preprocesamiento en el cual consiste en la concatenación de caracteres dentro del archivo
InstructionsFileFunction()	Obtiene el nombre del archivo de instrucciones llamado "metainstance.txt" mediante una lectura en base de tokens, en el archivo "logbook.txt"
InstancesNamesFunction()	Obtiene el nombre de la instancia o instancias con las que se trabajara mediante la lectura en base de tokens
InstancesFunctions()	Obtiene la cantidad de instancias ignorando las cadenas y los paréntesis mediante una lectura en base de tokens, en el archivo "logbook.txt"
CompileLogbook()	Permite realizar la compilación de archivo "logbook.txt". este proceso consta de los siguientes pasos: 1.-Separar las instrucciones (separador = ";") 2.- Leer cada instrucción hasta el paréntesis apertura 3.- En base a la instrucción leída, verificar que sea una del diccionario 4.- Si es una palabra reconocida por el diccionario, entonces: 4.1.- Obtener el tipo de argumento a ser leído y almacenarla en la variable correspondiente 4.2.- si no, mostrar el mensaje de error correspondiente y salir
GetMetainstance()	Permite leer el archivo "metainstance.txt" mediante una lectura en base de tokens e identificando la cantidad de variables checando los símbolos, saltos de línea y palabras reservadas para comprobar que están en el diccionario. Una vez que el sistema encuentre un símbolo ";" comprueba si es la última instrucción y si no lo es, guarda la variable y el tipo de variables de la siguiente línea en el sistema para su uso posterior.
CalculateNumberOfVaria-bles()	Método mediante el cual obtiene la cantidad de variables del archivo "metainstance.txt", que se especifica en la primera línea, checa si la instrucción contiene la palabra reservada "variables" entonces se obtiene el valor que se encentra dentro de los tokens "("y ")" y lo devuelve como un entero.
ResizeStructures()	Método en el cual se redimensionan las estructuras de almacenamiento en las nuevas dimensiones obtenidas para el problema de empaquetado, en base de los métodos: 1.- Func_instancias() 2.- CalculaCantVariables()
LoadInstance()	Método por el cual se cargan las instancias especificadas en el archivo "logbook.txt" a las estructuras de control dentro de la herramienta aplicado para el problema de empaquetado.
ShowInstances()	Método en el cual se muestra en pantalla las instancias que han sido cargadas.
Main()	Método principal en el cual se llaman una serie de funciones que tiene como función el preprocesamiento de la herramienta
GetAlgortihm()	Obtiene el nombre del algoritmo y prepara el objeto algorithm
GetPopSize()	Obtiene el valor del tamaño de la población
GetMaxIter()	Obtiene el valor de las generaciones máximas
GetOperator()	Obtiene el nombre del operador y prepara el objeto operator
GetPercentage()	Obtiene el valor del porcentaje de ocurrencia

Los métodos resaltados en la Tabla 5 fueron propuestos debido a que cumplen con la función de guardar la configuración de los experimentos computacionales, además de que reutilizando los métodos referentes a logbook se hizo más sencillo el trabajar con las diferentes instancias de los casos de estudio.

3.2 Metodología del módulo de comunicación

En esta sección se presenta la metodología propuesta para el proceso de comunicación entre VisTHAA y M-SDOSS tomando como base la metodología propuesta en el trabajo de Eureka Universe, dicho trabajo ha sido probado para comunicación escritorio/API.

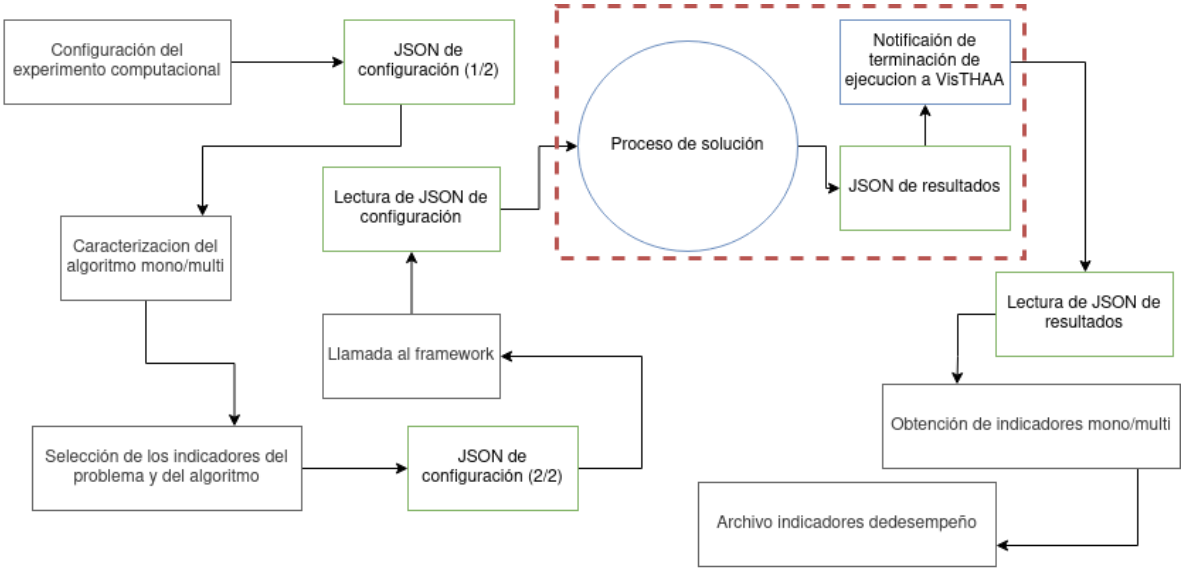


Figura 35 Metodología del proceso de comunicación de VisTHAA/M-SDOSS

En la Figura 35 se muestra la metodología a seguir para la comunicación entre VisTHAA/M-SDOSS, el cual está conformado por una serie de fases y elementos, las cuales se explican a lo largo de este trabajo de tesis.

Los elementos necesarios para llevar a cabo la ejecución de la metodología mostrada en la Figura 36 son: la configuración del experimento computacional, los indicadores de desempeño mono/multi objetivo y la cantidad de ejecuciones del experimento computacional, ejecución de experimento y cálculo de indicadores de desempeño. A continuación, se da una descripción de las fases involucradas en la metodología del proceso de comunicación.

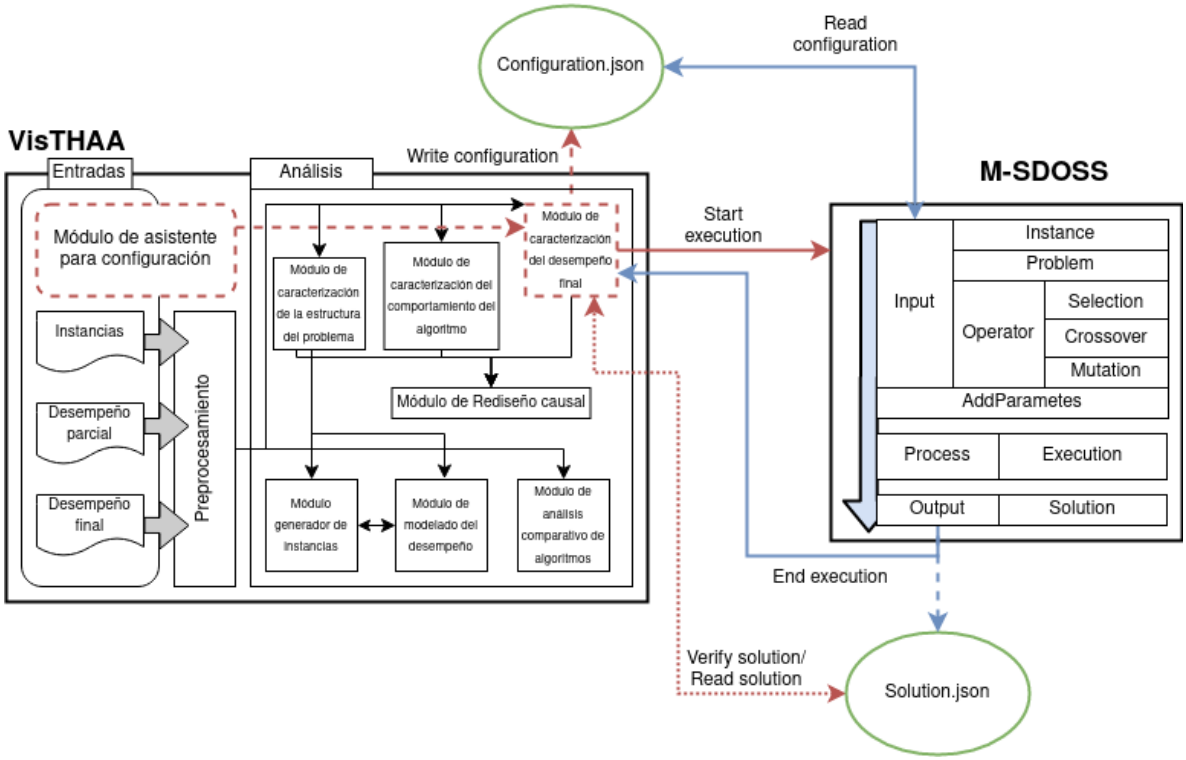


Figura 36 Diagrama de proceso de comunicación VisTHAA/M-SDOSS

3.2.1 Configuración del experimento computacional

Dentro de este apartado se busca apoyar al usuario en la construcción de experimentos computacionales, con respecto al siguiente formato:

1. Selección de problema a resolver (mono objetivo: mochila | multi objetivo: PSP y DTLZ).
2. Selección del logbook (Un archivo de texto plano que contiene la cantidad de instancias que se desea cargar, el patrón de la instancia y la lista de nombres de cada instancia).
3. Selección de parámetros del algoritmo solucionador (tamaño de la población y cantidad de generaciones).
4. Selección de operadores genéticos y sus parámetros (probabilidad de frecuencia).

Cabe aclarar que se planteó en un principio dejar una selección de algoritmo libre pues el framework lo permite, pero debido a que VisTHAA actualmente tiene asociados los problemas con los algoritmos solucionadores no fue posible agregar una configuración libre y se mantuvo la existente.

Este es el primer paso para generar los objetos: experiment, algorithm, problem, operator, etc. Estos son usados para generar el archivo de configuración en formato JSON.

3.2.2 Selección de indicadores de desempeño

Una vez seleccionada la configuración del experimento se requiere la selección de una instancia específica para terminar la generación del JSON de configuración.

Además, es en este punto donde serán seleccionados los indicadores deseados para el análisis, así como la cantidad de ejecuciones que se desea realizar del experimento computacional.

3.2.3 Ejecución del experimento

Para realizar la ejecución del experimento dentro del framework, VisTHAA genera el archivo JSON de configuración con las especificaciones previstas y realiza la petición de ejecución sobre el framework M-SDOSS (.jar) a forma de aplicación externa, como se comentó anteriormente en la subsección de preparación ejecuta un main genérico que busca el archivo de configuración en una ruta especificada y luego realiza el experimento computacional con respecto al JSON de configuración.

Cabe aclarar que VisTHAA realizara las n peticiones de ejecución especificadas por el usuario, esto es la cantidad de veces que se ejecutara él .jar del framework. de tal manera que el resultado por parte de estas ejecuciones será la escritura de n archivos JSON de solución que VisTHAA verificará con respecto a la cantidad de ejecuciones solicitadas.

3.2.4 Cálculo de indicadores

Comprobado que todas las ejecuciones han sido resueltas, las soluciones son guardadas y enviadas al proceso de análisis el cual es realizado internamente por VisTHAA.
En la siguiente actividad se muestran los resultados obtenidos de algunas de las pruebas hechas, que contrastan los valores obtenidos por el algoritmo interno de VisTHAA y el algoritmo de M-SDOSS.

3.2.5 Parámetro de medida

Dando seguimiento al comentario del comité, esta última parte de la actividad se presentan las alternativas para medir el rendimiento/efectividad del proceso de comunicación entre VisTHAA y M-SDOSS.

Las alternativas propuestas se presentan como capaz que es están presenta en procesos de conexión de red, estas capaz se entienden como subprocesos que funcionan de manera similar dentro del proceso de comunicación propuesto para este proyecto.

3.2.5.1 Capa de conexión:

Dentro del proceso de comunicación actual esta capa estaría abarcando desde que VisTHAA realiza las peticiones de ejecución al framework hasta la validación del término de las ejecuciones, el proceso está marcado por flechas color rojo en la Figura 36.

3.2.5.2 Capa de visualización:

El proceso de visualización del protocolo de comunicación propuesto se está definido en dos partes, la primera siendo realizada mientras se resuelve cada proceso de ejecución individual, cada una de las peticiones es mostrada en consola. Mientras que la segunda es al término del proceso total de ejecuciones y genera el análisis, es aquí donde se despliega un mensaje avisando que el proceso ha terminado y la ruta donde se han colocado los resultados, marcado por flechas color azul en la Figura 36.

Capítulo 4 Experimentación y análisis de resultados

En este capítulo se muestran los resultados obtenidos de la aplicación de las metodologías y procesos propuestos en el Capítulo 3, aplicado la implementación del módulo de asistencia para la configuración del experimento computacional y el rediseño de los módulos de caracterización del desempeño final aplicado a problemas de optimización mono y multi objetivo. Como contribución se incorporó a VisTHAA y a M-SDOSS los submódulos de conexión y manipulación de JSON, para el aumento de su funcionalidad.

4.1 Ambiente experimental

En esta sección se muestran las características del equipo experimental, las herramientas de software que se emplearon, así como explicaciones sobre cada una de las partes que componen los casos de estudio propuestos en el Capítulo 2 para demostrar el cumplimiento de este trabajo de tesis. En la Tabla 9 se muestran las características del equipo en donde se realizaron las experimentaciones y los programas utilizados para el desarrollo del para desarrollar el asistente de configuración de experimentos computacionales y la conexión dentro de los procesos de caracterización de rendimiento en sus versiones mono y multi objetivo.

Tabla 9 Características del equipo experimental.

Hardware	**Software**
Laptop, intel core i7-7700HQ CPU@2.80GHz, 8Gb de RAM, S.O. Windows 10 Home Professional x64, y Pop! _OS20.10 x64.	• NetBeans 8.1 y 12.0 • InteliJ 2020.3.2

En la Tabla 10 se describen las características del conjunto de instancias pertenecientes a los problemas utilizados en los casos de estudio.

Tabla 10 Características del conjunto de problemas para casos de estudio.

Problema	**Objetivo**	**Numero de instancias**	**Características**	
Mochila	Mono	4	Numero de objetos	15,24,100,1000
			Capacidad de la mochila	750,6404180,3254,13001
Selección de portafolio de cartera de proyectos	Multi	3	Presupuesto máximo	250000
			Numero de objetivos	9

			Numero de áreas	2
			Numero de regiones	3
			Número de proyectos	100
DTLZ 1	Multi	1	Numero de variables	7
			Objetivos	3
DTLZ 2	Multi	1	Numero de variables	12
			Objetivos	3
DTLZ 3	Multi	1	Numero de variables	12
			Objetivos	3

La Tabla 11 especifica las configuraciones del conjunto de algoritmos solucionadores utilizados en el caso de estudio para verificación de comunicación.

Tabla 11 Características de conjunto de algoritmos solucionadores.

Algoritmo solucionador	**Parámetros**	**Valores**
Genético	Tamaño de la población	100
	Numero de generaciones	1000
	Porcentaje de cruza	70%
	Porcentaje de muta	5%
NSGA-II	Tamaño de la población	100
	Numero de generaciones	1000
	Porcentaje de cruza	70%
	Porcentaje de muta	5%

Para los experimentos mostrados en las secciones posteriores se realizó una selección de los siguientes indicadores para experimentación 1, 2 y 3: [Agregar los indicadores seleccionados]

Los experimentos tienen la finalidad de presentar el funcionamiento completo del módulo de comunicación entre VisTHAA y M-SDOSS validando la capacidad de realizar análisis de desempeño de algoritmos que no se encuentran dentro de VisTHAA y debido a que el proceso de ejecución del algoritmo se realiza íntegramente en el framework conectado a la herramienta de análisis.

4.2 Experimentación 1: desempeño AG solucionando problema de la mochila

En esta sección se muestran los resultados obtenidos de la experimentación del proceso de caracterización de desempeño sobre el algoritmo genético existente en la Herramienta M-SDOSS aplicado al problema de la mochila mono objetivo, el cual pertenece a la familia de contenedores.

La Figura 37 describe la interfaz principal de la herramienta VisTHAA, con la incorporación de los nuevos módulos propuestos en esta tesis. Remarcado en un cuadro rojo se denotan los módulos que se explicaran a lo largo de este capítulo.

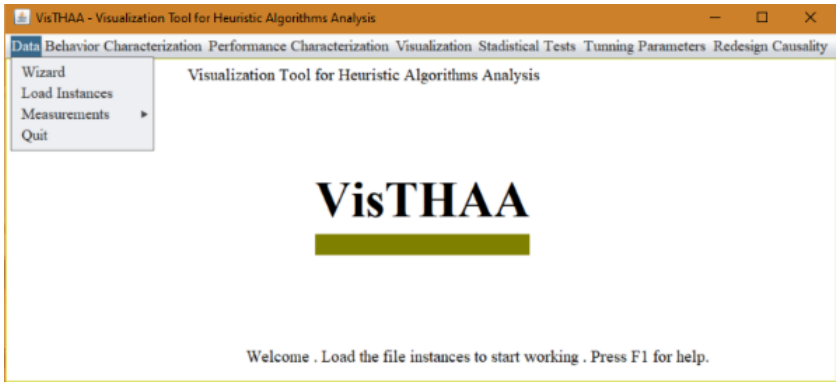


Figura 37 Ventana principal de VisTHAA

En este trabajo se planteó la incorporación de un módulo que facilite al investigador la selección de la configuración del experimento computacional a analizar, en la Figura 38 se presenta la interfaz principal del módulo asistente de configuración propuesto en esta tesis. En el algoritmo 4 se describe el funcionamiento interno del módulo mencionado.

Algoritmo 4 Proceso del módulo *ConfigurationWizard.*

1. **Inicio**
2. **Selección** del problema
3. **Búsqueda** de la selección del archivo logbook (Conjunto de instancias)
4. **Si** el tipo de instancia es igual al problema
5. **Ejecución** del método de *preprocesamientoInstacia*
6. **Sino**
7. Archivo logbook no pertenece al tipo de problema a procesar
8. **Selección** del tamaño de la población
9. **Selección** de la cantidad de generaciones máxima
10. **Selección** de operador de selección
11. **Selección** de operador de cruza
12. **Selección** de operador de muta
13. **Ejecución** del proceso de formato para el archivo de configuración p1
14. **Guardado** del objeto algoritmo
15. **Guardado** del nombre del algoritmo -> *name*
16. **Guardado** del tamaño de la poblacion -> *populationSize*
17. **Guardado** del número maximo de generaciones -> *iterationMax*
18. **Guardado** del objeto operador de cruza
19. **Guardado** del nombre del operador -> *name*
20. **Guardado** del porcentaje de ocurrencia -> *probability*
21. **Guardado** del objeto operador de muta
22. **Guardado** del nombre del operador -> *name*
23. **Guardado** del porcentaje de ocurrencia -> *probability*
24. **Guardado** del objeto operador de selección
25. **Guardado** del nombre del operador -> *name*
26. **Guardado** del tamaño del torneo -> *size*
27. **Guardado** del problema -> *problemType*
28. **Guardado** de datos adicionales -> *numberOfVariables, NumberOfObjetives, NumberOfConstraints*
29. **Selección** de indicadores de desempeño
30. **Ejecución** de proceso de guardado de indicadores
31. **Guardado** de lista de indicadores -> *metricNames*
32. **Selección** de cantidad de ejecuciones
33. **Ejecución** del proceso de formato para el archivo de configuración p2
34. **Guardado** de la cantidad de ejecuciones -> *runNumber*
35. **Ejecución** del proceso de generación de archivo de configuración
36. **Escritura** del archivo *configuration.json*
37. **Fin**

El algoritmo 4 inicia con la petición de la selección del problema e instancia (Líneas 2-7). Una vez especificados dichos elementos, se procede a obtener los atributos de la configuración del algoritmo (tamaño de la población, generaciones máximas, operador de selección, operador de muta y operador de cruza | líneas 8-12) y se procede a realizar el guardado/procesamiento de los distintos atributos en el objeto configuración.json (Líneas 13-28).

Por último, se realiza la selección de los indicadores para el experimento computacional, así como la instancia sobre la instancia que será enviada, además de la cantidad de ejecuciones en caso de tratarse de caracterización mono-objetivo (Líneas 29-35). Una vez seleccionados los últimos atributos se agregan los datos de cantidad de experimento y los valores de la instancia

seleccionada al objeto configuración.json para crear el archivo configuración.json(Líneas 35 y 36)

En el siguiente panel se presenta un listado de problemas y un botón con el que se guardan la lista de instancias que se podrán procesar más adelante.

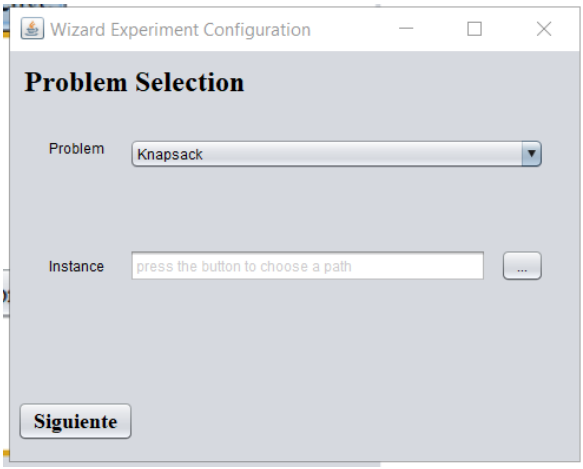


Figura 38 Modulo de ConfigurationWizard

El proceso que desata el botón es el siguiente, se inicia con la búsqueda del archivo logbook el cual contiene los nombres del conjunto de instancias a analizar y la referencia al archivo metainstance. En la Figura 39 se muestra la ventana de búsqueda del archivo logbook.

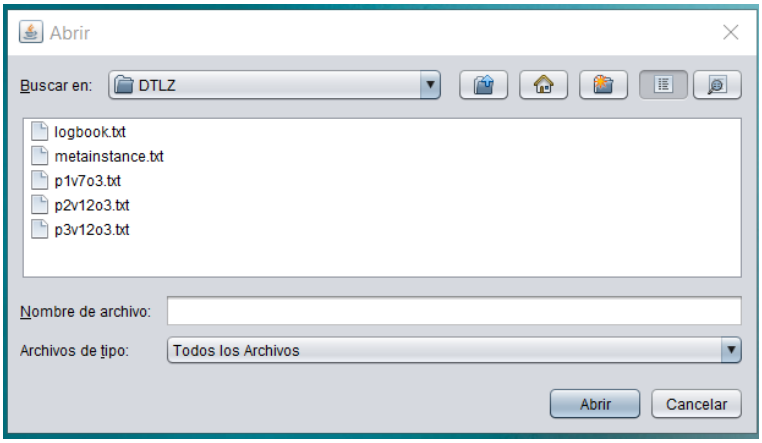


Figura 39 Ventana de búsqueda del archivo logbook

Después de la selección del archivo logbook, se procesa la instancia y se preparan la ruta para la lectura de las instancias que contiene el archivo. A continuación, las Figuras 40, 41, 42 muestran ventanas indicando la carga de los diferentes problemas debido a que este proceso es similar en cada uno de los experimentos se optó por mostrar todo el proceso solamente el primer experimento.

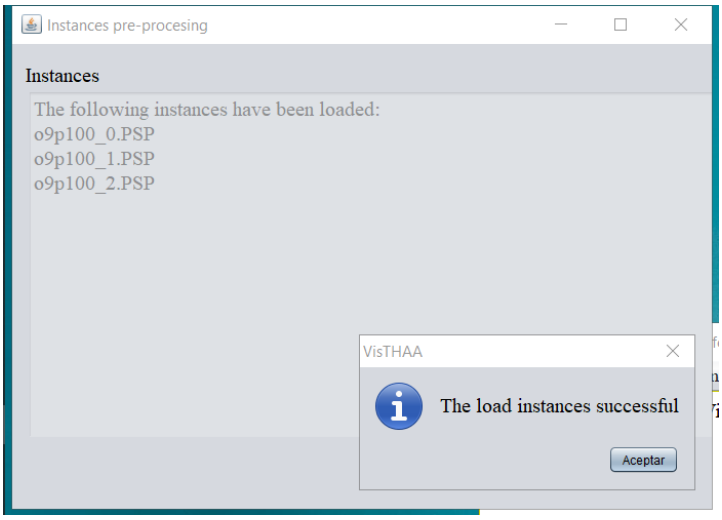


Figura 40 Ventana de instancias cargadas del Problema de proyectos

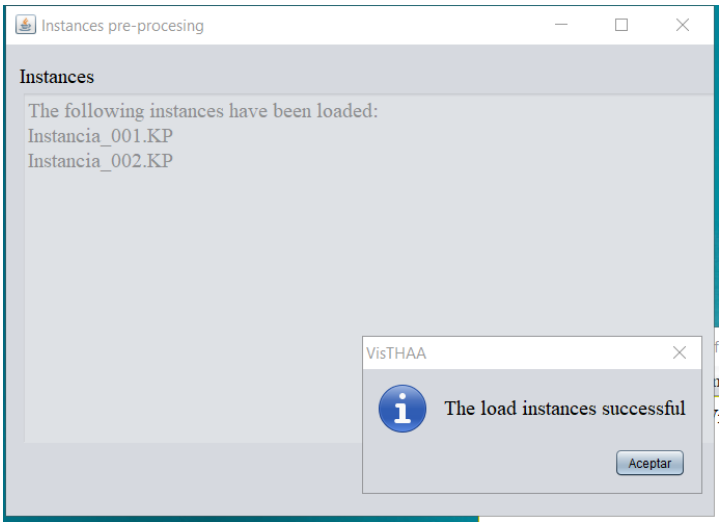


Figura 41 Ventana de instancias cargadas del problema de la mochila

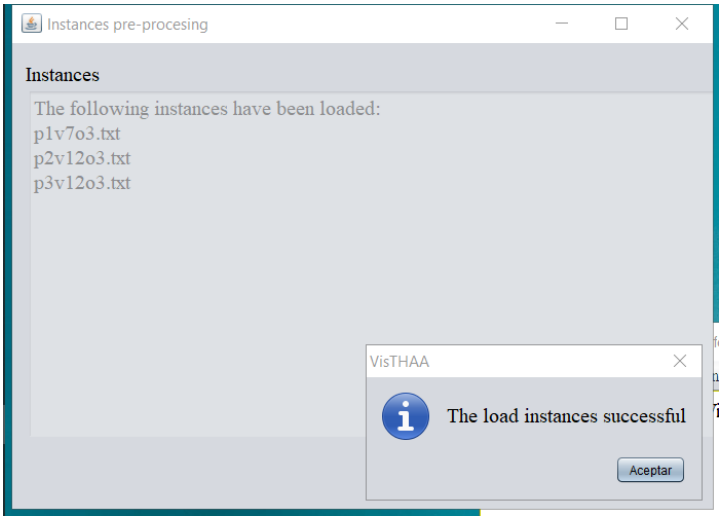


Figura 42 Ventana de instancias cargadas de DTLZ

Después como se puede observar en la Figura 41 y 42, están la selección de atributos del algoritmo general y la selección de operadores para el algoritmo respectivamente.

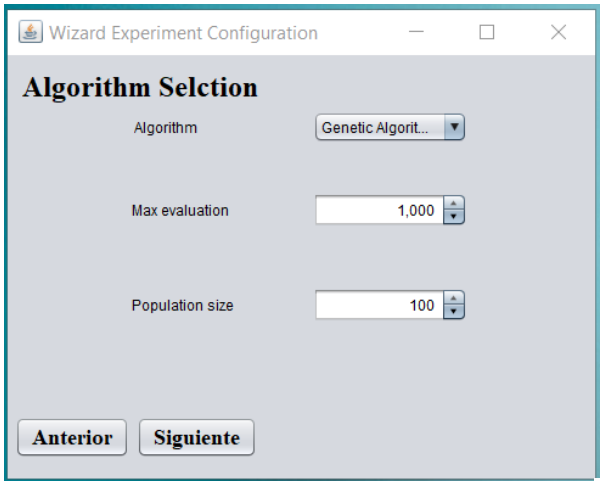


Figura 43 selección de atributos de algoritmo ConfigurationWizard

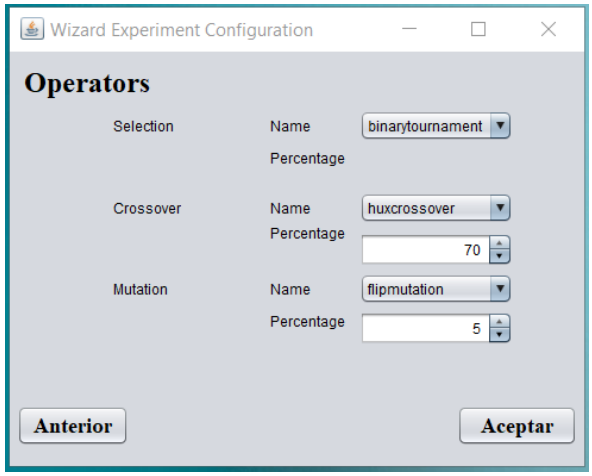
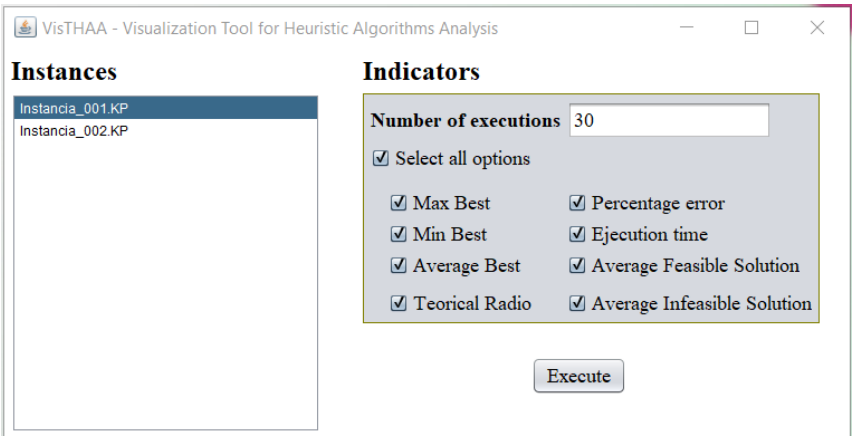


Figura 44 selección de operadores ConfigurationWizard

Este proceso es idéntico para cada uno de los experimentos, cabe destacarse con respecto a la Figura 43 que la parte del módulo que permite cambiar de algoritmos es definida por el tipo de problema que se desea resolver siendo que se trabaja con genético en caso de ser mono/objetivo o será NSGA-II en caso de ser multi-objetivo, esto debido a las limitaciones existentes dentro de VisTHAA para permitir la generalización de estos métodos para más información revisar el Anexo B con respecto a las modificaciones realizadas para problemas mono-objetivo.

A partir de este punto se muestra la ventana de selección de instancia, indicadores y el número de ejecuciones en la Figura 45, donde se aprecia en el lado izquierdo de la Figura la lista de instancias encontradas en el logbook, la posibilidad de teclear la cantidad de ejecuciones y permitir entre la variedad de indicadores posibles o la selección de todos en caso de así requerirlo.

En algoritmo 5 se describe el proceso interno del módulo Performance mono objetive/ Conection



que fue modificado para permitir la comunicación con M-SDOSS.

Figura 45 Módulo de Performance mono-objetive

Algoritmo 5 Proceso de Caracterización del Desempeño mono-objetivo y conexión a Framework

1. **Inicio**
2. **Selección** del conjunto de instancias a caracterizar
3. **Selección** del conjunto de índices a utilizar
4. **Especificar** el número de ejecuciones
5. **Llamar** el proceso de caracterización del módulo Performance Mono Objetive
6. **Guardar** el número de ejecuciones → *numEjecuciones*
7. **Guardar** el número de instancias a caracterizar → *numInstancias*
8. **Ciclo** *i*=0 a *numInstancias*
[modificar]
9. resultadosDesempeño [*numEjecuciones*]
10. **Ciclo** *j*=0 a **numEjecuciones**
11. Obtención del desempeño
12. Guardar el desempeño → resultadosDesempeño[j]

El Algoritmo 12 inicia con la selección del conjunto de instancias que se desean caracterizar, el número de ejecuciones que el investigador desea y la selección del conjunto de índices que se aplicaran al algoritmo (Líneas 2-4). Una vez especificadas las instancias con las que se desea analizar, el número de iteraciones y los índices a aplicar al algoritmo, se guarda cada atributo en variables para su uso posterior y se inicia el proceso de caracterización (Línea 5-7).

El módulo trabaja de la siguiente manera, se ejecuta el algoritmo n cantidad de veces por instancias y en cada iteración se aplica la fórmula de los índices seleccionados y se guarda cada resultado para obtener un valor promedio por índice seleccionado (Líneas 8-16).

Una vez el proceso es terminado, para todo tipo de experimentación se mostrará una ventana emergente como la mostrada en la Figura 46, donde se da a conocer que el proceso finalizo de la manera esperada y es mostrada la ruta donde son escritos los resultados de los indicadores seleccionados.

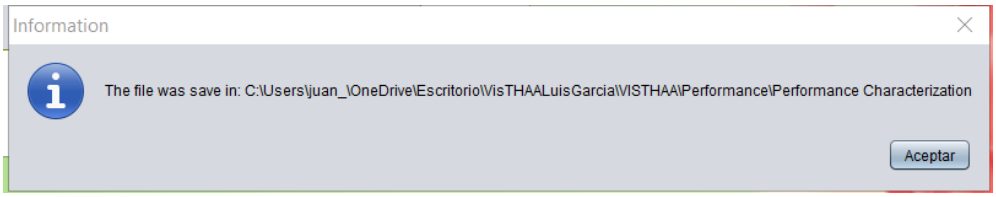


Figura 46 Ventana emergente de finalización y escritura de resultados de caracterización del desempeño

4.2.1 Resultados del desempeño AG solucionando problema de la mochila

Este experimento se utiliza para validar la comunicación entre las dos herramientas (VisTHAA y M-SDOSS), se utilizan en este primer experimento un problema mono objetivo que puede ser encontrado dentro de VisTHAA.

La Tabla 12 muestran los resultados para los indicadores de desempeño siguientes: mejor máximo, mejor mínimo, promedio mejor, radio teórico, porcentaje de error, tiempo de ejecución, promedio de soluciones factibles y promedio de soluciones infectables.

Tabla 12 Resultados de análisis de desempeño por algoritmo genético para problema a de la mochila de M-SDOSS

Instancia	Max B	Min B	AB	TR	PE	TE	AFS	AIS
Instancia10.KP	679	679	679	2.197411003	-1.197411003	888	0.34	0.66
Instancia15.KP	2756	2756	2756	1.890260631	-0.890260631	1350	0.02	0.98
Instancia24.KP	2.46E+07	2.46E+07	2.46E+07	1.812226043	-0.812226043	799	0.01	0.99
Instancia100.KP	4625	4625	4625	0.939658675	0.060341325	1232	0.01	0.99
Instancia1000.KP	21323	21323	21323	1.452025877	-0.452025877	1250	0.01	0.99

Los resultados fueron revisados contra una comparación de los indicadores obtenidos contra el algoritmo interno de VisTHAA, para de esta manera comprobar 2 cosas: primero que el funcionamiento del proceso sea correcto y segundo que los resultados obtenidos por M-SDOSS se pudieran representar para ser analizados.

La única desventaja debido al comunicación entre VisTHAA y M-SDOSS es que para instancias pequeñas tarda mucho pero conforme la instancia es mayor, el proceso en cuento a tiempo se estabiliza.

Con este experimento se comprueba que existe comunicación mediante la metodología propuesta en su fase mono-objetivo, ya que de VisTHAA fue escrita la configuración del experimento computacional (AG para solución de problema de la mochila) y se estableció contacto con el framework este proceso la configuración, ejecuto los experimentos y escribió los resultados del experimento, los cuales VisTHAA proceso para realizar el cálculo de los indicadores mostrados en la Tabla 12.

4.3 Experimentación 2: desempeño NSGA-II solucionando PSP

Los pasos anteriores que mostrar modulo performace mono-objetive siguen el mismo desarrollo para todos los experimentos por lo que serán omitidos, y se empezara mostrando la Figura 47 donde se puede apreciar en el lado izquierdo la selección de la instancia a trabajar, seguido del área de selección de indicadores de desempeño multi-objetivo donde se tiene la opción de seleccionar algunos en concreto o todos en caso de así requerirlo, por parte del experimento realizado los únicos indicadores no seleccionados son NS, NW y NF pues no realizaban ningún proceso para el algoritmo NSGA-II para más información al respecto revisar el apéndice C en donde se realiza una inspección a los procesos y métodos que realiza este módulo.

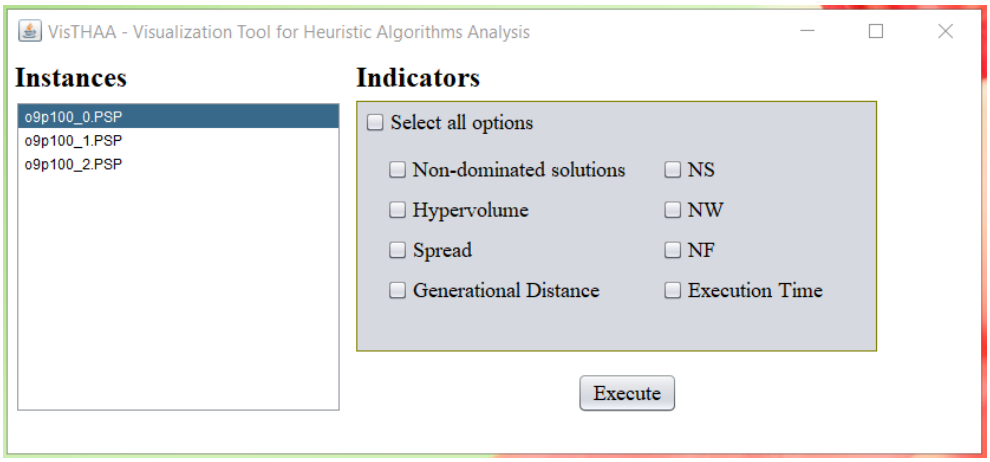


Figura 47 Modulo performance multi objetive – psp intances

En el algoritmo 6 se describe el proceso interno del módulo Performance mono objetive/ Conection que fue modificado para permitir la comunicación con M-SDOSS, este proceso es el mismo tanto para este experimento como para el siguiente así que será omitido a posteriori.

Algoritmo 6 Proceso de Caracterización del Desempeño multi objetivo y conexión a Framework

1. **Inicio**
2. **Selección** del conjunto de instancias a caracterizar
3. **Selección** del conjunto de índices a utilizar
4. **Llamar** el proceso de caracterización del módulo Performance
Multi Objetive
5. **Guardar** el número de instancias a caracterizar → *numInstancias*
6 **Ciclo** *i*=0 a *numInstancias*
7. **Obtención** del desempeño
8. **Si** el índice esta seleccionado
9. **Aplicar** índice y guardar su resultado
10. **Fin** condición si
11. **Fin** ciclo *i*
12. Fin

El Algoritmo 13 inicia con la selección del conjunto de instancia que se desean caracterizar y el conjunto de índices que el investigador desea aplicar al desempeño del algoritmo (Líneas 2-3), después se inicia el proceso con la ejecución del algoritmo por cada una de las instancias seleccionadas obteniendo su desempeño y por último se aplican los índices seleccionados al resultado obtenido del algoritmo (Líneas 4-11)

Una vez terminado el proceso de caracterización mono-objetivo o multi-objetivo, la herramienta VisTHAA guarda los resultados obtenidos de los índices seleccionados a un archivo de texto plano (.txt), el cual servirá como información de entrada para el módulo de visualización gráfica.

4.2.1 Resultados del desempeño NSGA-II solucionando PSP

Este experimento se utiliza para profundizar en la validación del proceso de comunicación entre VisTHAA y M-SDOSS, esta vez utilizando un problema multi-objetivo que de igual forma está presente en VisTHAA, pero permitió descubrir las implicaciones que existen dentro del módulo de caracterización del rendimiento en su fase multi-objetivo.

La Tabla 13 muestran los resultados para los indicadores de desempeño siguientes: Soluciones no dominadas, hipervolumen, propagación, distancia generacional y tiempo de ejecución.

Tabla 13 Resultados de análisis de desempeño por algoritmo NSGA-II para PSP de M-SDOSS

Instancia	Non-dominated_Solutions	Hypervolume	Spread	Generational_Distance	Execution_Time
o9p100_0.PSP	1	NaN	NaN	0	1.017
o9p100_1.PSP	1	NaN	NaN	0	1.085
o9p100_2.PSP	1	NaN	NaN	0	0.984

Para los resultados de este experimento fue necesario agregar un proceso al momento de ejecución dentro de M-SDOSS debido a que el indicador de distancia generacional requiere obtenerse mientras se ejecuta el experimento computacional, así que se agregó el proceso en caso de ser requerido M-SDOSS ser hace cargo el calcular la distancia generacional y devolverla como parte de la solución,json (Archivo de resultados) donde VisTHAA estaría preparado para procesar el resto de la información y mostrar los resultados finales, para más información al respecto revisar el anexo A donde se presenta la librería para manipulación de JSON y la definición de los objetos de M-SDOSS.

Con este experimento se comprueba que existe comunicación mediante la metodología propuesta en su fase multi objetivo, ya que de VisTHAA fue escrita la configuración del experimento computacional (NSGA-II para resolver PSP) y se estableció contacto con el framework este proceso la configuración, ejecuto los experimentos y escribió los resultados del experimento, los cuales VisTHAA proceso para realizar el cálculo de los indicadores mostrados en la Tabla 13.

4.4 Experimentación 3: desempeño NSGA-II solucionando DTLZ's

Los pasos anteriores que mostrar módulo performace mono-objetive siguen el mismo desarrollo para todos los experimentos por lo que serán omitidos, y se empezará mostrando la Figura 48 donde se puede apreciar de igual manera las secciones mencionadas en el anterior experimento a diferencia de las instancias implicadas, en este caso se cuenta con los problemas DTLZ.

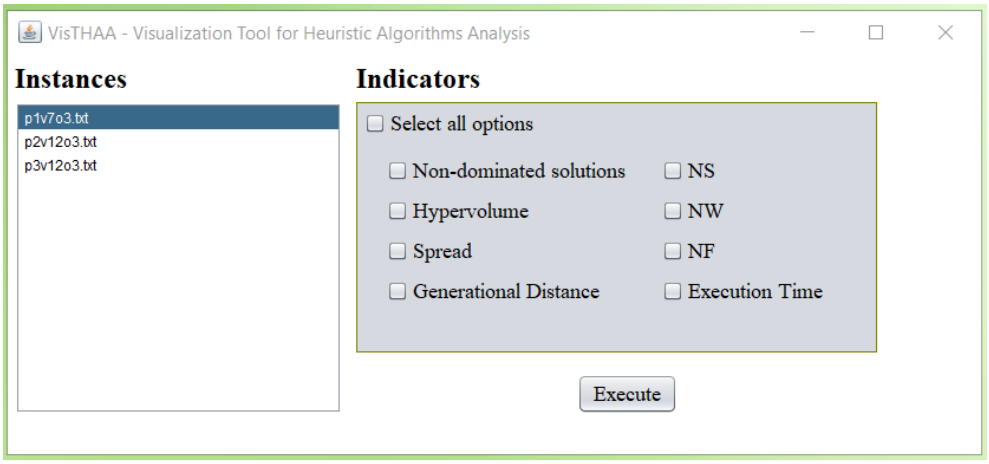


Figura 48 Modulo performance multi objetive – dtlz intances

4.2.1 Resultados del desempeño NSGA-II solucionando DTLZ's

Este experimento se utiliza para demostrar que el proceso de comunicación permite a VisTHAA acceder a problemas definidos dentro del framework y que actualmente no existen dentro de VisTHAA, siendo el caso los DTLZ 1, 2 Y 3.

La Tabla 14 muestran los resultados para los indicadores de desempeño siguientes: Soluciones no dominadas, hipervolumen, propagación, distancia generacional y tiempo de ejecución.

Tabla 14 Resultados de análisis de desempeño por algoritmo NSGA-II para DTLZ de M-SDOSS

Instancia	Non-dominated_Solutions	Hypervolume	Spread	Generational_Distance	Execution_Time
p1v7o3.txt	1	9258.770305	1	0.15701245	2.604
p2v12o3.txt	1	9259.44961	1	0.00949211	1.054
p3v12o3.txt	1	9256.752023	1	4.991709033	0.728

Para los problemas DTLZ fue necesaria la modificación de algunos de los métodos que permiten procesar los resultados debido a que similar a lo comentado con anterioridad en el segmento de resultados de PSP, se presentó el problema de que la distancia generacional debía ser calculada al momento de ejecución, por lo que se tomó la decisión de que efectivamente en caso de ser seleccionado el indicador, seria trabajo del M-SDOSS llevar a cabo este proceso, cuyo resultado es de vuelto dentro del solución.json, presentando se cómo una validación indirecta de que el proceso del comunicación se lleva acabo de manera adecuada.

Con este experimento se comprueba que existe comunicación mediante la metodología propuesta en su fase multi objetivo, ya que de VisTHAA fue escrita la configuración del experimento computacional (NSGA-II para resolver DTLZ) y se estableció contacto con el

framework este proceso la configuración, ejecuto los experimentos y escribió los resultados del experimento, los cuales VisTHAA proceso para realizar el cálculo de los indicadores mostrados en la Tabla 14, además de corroboran que una vez definido el objeto de problema o algoritmo del framework, VisTHAA es capaz de generar una configuración que permita su uso a pesar de no estar presente en su repositorio interno.

Capítulo 5 Conclusiones y trabajos futuros

En este capítulo se presenta a manera de conclusión, las aportaciones realizadas en este trabajo de investigación y la posible continuidad a futuro para los siguientes investigadores.

5.1 Conclusiones

Las conclusiones a las que se llegaron en esta labor fueron las siguientes:
Se cumplio con el objetivo principal de pemitir la comunicacion entre VisTHAA y M-SDOSS mediante un módulo de comunicación permitiendo asi que los experimentos computacionales realizados en el framework se puedan ser analizados en cuanto a desempeño por VisTHAA. Dicho objetivo requirio de la implementación de un asistente de configuración que permitiera la manipulación de los atributos que el framework definia para su experimentos computacionales, ademas de la revision y modificacion de algunas partes del modulo de caracterización de rendimiento, especificamente hablando en los procesos de ejecucion de los algoritmos para permitir realizar peticion a framework como un programa ajeno a VisTHAA.

La validación del proceso de comunicación fue mostrado mediante los diferentes casos de estudio, de donde se obtuvieron los resultados de diferentes experimentos computacionales, ademas de que el proceso de el indicador de distancia generacional requeria ser calculado en el momento de la ejecucion por lo que es un indicador que se calcula en el M-SDOSS pero es visualizado gracias a VisTHAA.

Las principales contribuciones son las siguientes:

1. Diseño de una metodologia general para el procesamiento de las configuraciones de los experimentos computacionales.
2. Diseño de una metodologia general para la comunicacion entre VisTHAA y un framework de optimizacion.
3. Rediseño de proceso de ejecución de algoritmos en el modulo de caracterización de desempeño mono y multi objetivo.
4. Incorporación de los problemas y algoritmos de M-SDOSS para ser analizados en VisTHAA.
5. Rediseño del algoritmo genetico de VisTHAA.

5.2 Trabajos futuros

En esta sección se sugieren los siguientes trabajos futuros y líneas de investigación que complementarían la herramienta VisTHAA, las cuales se mencionan a continuación:

1. Rediseño del módulo de entrada para nuevas instancias a analizar.
2. Rediseño de los módulos de caracterización del desempeño y comportamiento para los nuevos problemas y algoritmos a estudiar.
3. Incorporación de más algoritmos y operadores genéticos para la configuración de los experimentos computacionales.
4. Rediseño e implementación de la interfaz de usuario de VisTHAA.
5. Implementación de una versión web de VisTHAA.
6. Revisión y Rediseño de los algoritmos solucionadores de VisTHAA.

Referencias

.

[Álvarez, 2006] Álvarez, V. (2006). Modelo para representar la Complejidad del problema y el desempeño de algoritmos. Tesis de maestría. Instituto Tecnológico de Cd. Madero, Tamaulipas, México.

[Alcalá-Fdez, 2011] Alcalá-Fdez, J., Fernández, A., Luengo, J., Derrac, J., García, S., Sánchez, L., & Herrera, F. (2011). Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. Journal of Multiple-Valued Logic & Soft Computing, 17.

[Balderas, 2012] Balderas, F. (2012). Sistema de Apoyo a la Decisión para la Selección de Cartera de Proyectos en Organizaciones Públicas. Tesis de maestría. Instituto Tecnológico de Ciudad Madero, Tamaulipas, México.

[Balderas, 2018] Balderas, F., Fernández, E., Gómez, C., Cruz, L., Rangel, N., Morales, M. (2018). A grey mathematics approach for evolutionary multi-objective metaheuristic of project portfolio selection, Springer, Cham, 379-388.

[Bisdorff, 2015] Bisdorff, R., Dias, L. C., Meyer, P., Mousseau, V., & Pirlot, M. (2015). Evaluation and Decision Models with Multiple Criteria.

[Castillo, 2011] Castillo, N. (2011). Evaluación de Estrategias de Mejora de desempeño de metaheurísticas aplicados a BBP Vía Diagnóstico Visual. Tesis de maestría. Instituto Tecnológico de Ciudad Madero, Tamaulipas, México.

[Cruz, 2004] Cruz, L. (2004). Caracterización de Algoritmos Heurísticos Aplicados al Diseño de Bases de Datos Distribuidas. Tesis de doctorado. Centro Nacional de Investigación y Desarrollo Tecnológico, Cuernavaca, Morelos, México.

[Davies et al., 1979] Davies, D. et al. (1979). Computer Networks and Their Protocols; John Wiley & Sons; Chapter 6 - Communication Protocols and Interfaces.

[Deb, 2002] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. Evolutionary Computation, IEEE Transactions on, 6(2), 182-197.

[Deb, 2005] Deb, K., et al. (2005). Scalable Test Problems for Evolutionary Multiobjective Optimization. Springer.

[Diccionario L.E., 2020] Diccionario de la lengua española (febrero, 2020) Definición de protocolo. Recuperado de http://www.wordreference.com/definicion/protocolo.

[Dijkstra61] Dijkstra E. (1961). "ALGOL-60 Translation". Stichting Mathematisch Centrum. 2e Boerhaavestraat 49. Amsterdam. Rakenafdeling.

[Fernández, 2001] Fernández, E., & Navarro, J. (2001). Aplicación de metaheurísticas multiobjetivo a la solución de problemas de cartera de proyectos públicos con una valoración multidimensional de su impacto. Universidad Autónoma de Sinaloa.

[García, 2019] García, L. (2019). Modelado del desempeño de problemas NP-Duros aplicando análisis causal para la herramienta VISTHAA.

[Halim, 2006] Halim, S., Yap, Roland H. C.; and LAU, Hoong Chuin. Visualization for analyzing trajectory-based metaheuristic search algorithms. (2006). ECAI 2006: 17th European Conference on Artificial Intelligence, August 29-September 1, 2006, Riva del Garda, Italy. 141, 703-704. Research Collection School Of Information Systems.

[Haro, 2008] Haro, J. (2008). Diseño e implementación de un marco de trabajo (framework) de presentación para aplicaciones JEE.

[Holland, 1995] Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975).

[Json, 2020] Json (febrero, 2021). Introducción a JSON. Recuperado en febrero de 2021 de http://www.json.org/json-es.html.

[Martello, 1990] Martello, S., and Toth, P. Knapsack problems: algorithms and computer implementations. John Wiley & Sons, 1990.

[Martinez, 2017] Enith Martinez Cruz "Adaptación de un algoritmo de agrupación mono-objetivo a multi objetivo usando caracterización del desempeño".

[Mendenhall97] William, M., Terry, S. (1997) "Probabilidad y Estadística para Ingeniería y Ciencias". Fourth Edition. Prentice-Hall Hispanoamérica, S.A. ISBN (translation): 968-880-960-8. ISBN (original): 0-02-380581-1. (1997).

[Paradiseo, 2013] Humeau, Jérémie & Liefooghe, Arnaud & Talbi, El-Ghazali & Verel, Sebastien. (2013). ParadisEO-MO: From Fitness Landscape Analysis to Efficient Local Search Algorithms. Journal of Heuristics. 19. 10.1007/s10732-013-9228-8.

[Pérez, 2014] Pérez, M. (2014). Análisis de Algoritmos Metaheurísticos Vía Diagnóstico Estadístico (Tesis de Maestría), Instituto Tecnológico de Ciudad Madero.

[Quiroz, 2009] Marcela Quiroz Castellanos "Caracterización de Factores de Desempeño de Algoritmos de Solución De BPP" Instituto Tecnológico de Cd. Madero 2009.

[Rivero, 2010] Rivero, M. y Rabuñal, D., Dorado, J, Pazos, A. y Gestal, M. (2010). Introducción a los algoritmos genéticos y la programación genética. Universida de Coruña.

[Rojas, 2017] Rojas, R. (2017). Estudio del impacto de patrones de diseño en la implementación de un framework de optimización para apoyo a la toma de decisiones (Tesis de Maestría). Instituto Tecnológico de Ciudad Madero.

[Sánchez, 2016] Sánchez, L. (2016). Desarrollo de un protocolo de comunicación para un framework de apoyo a la toma de decisiones.

[Sánchez, 2007] Sánchez, P. (2007). Modelos para la combinación de preferencias en toma de decisiones: herramientas y aplicaciones. Tesis Doctoral, Universidad de Granada.

[Srinivas, 1994] N. Srinivas and Kalyanmoy Deb, Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms, Evolutionary Computation 2 (1994), no. 3, 221 – 248.

[Van, 1998] Van Veldhuizen, D. A., & Lamont, G. B. (1998). Multiobjective evolutionary algorithm research: A history and analysis. Technical Report TR-98-03, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

[Walsh, 1998] Walsh, N. (03 de octubre de1998). A Technical Introduction to XML. Recuperado de http://www.xml.com/pub/a/98/10/guide0.html.

[Wolper, 1995] Wolper, P. (1995). Computer aided verification. 1st ed. Berlin [u.a.]: Springer.

[YAML, 2001] YAML (2001). YAML Ain't Markup Language. Recuperado en febrero, 2020 de https://yaml.org.

[Zitzler99] Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength.

[Zitzler, 2000] Zitzler, E., Deb, K., & Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary computation, 8(2), 173-195.

.

Anexo A

Modulo/librería parser

En este anexo se presenta la librería que permitió tanto al framework M-SDOSS como a VisTHAA poder manipular el formato JSON, además de que a VisTHAA le permite tener la definición de los objetos de M-SDOSS para problemas, instancias, algoritmos y operadores.
Primero se presenta la parte de la librería que permite la manipulación de los formatos JSON y posterior mente se presentan brevemente las definiciones de cada uno de los objetos del experimento computacional definidos en M-SDOSS que VisTHAA utiliza.

A.1 Librería GSON

es un API en Java, desarrollada por Google, que se utiliza para convertir objetos Java a JSON (serialización) y JSON a objetos Java (deserialización). Se presentan ejemplos de serialización y deserialización con el fin de proporcionar conocimientos mínimos para entender la librería y los procesos de serialización y deserialización [7].

Esta librería estructura los JSON de la siguiente manera:

JsonElement: Esta clase representa cualquier elemento del Json que puede ser de alguno de los siguientes 4 tipos:
JsonObject: Esta clase representa un objeto en el Json; es decir, un conjunto de pares clave-valor donde las claves son strings y los valores son cualquier otro tipo de JsonElement.
JsonArray: Esta clase representa un array en el Json. Un array es una lista de JsonElements cada uno de los cuales puede ser de un tipo diferente. Se trata de una lista ordenada, por lo que el orden en que se añaden los elementos se conserva.
JsonPrimitive: Esta clase representa un tipo de dato primitivo u objetos de datos simples (String, Integer, Double, etc.).
JsonNull: Representa un objeto a null.

JsonObject, JsonArray, JsonPrimitive y JsonNull heredan de la clase JsonElement, como se representa en las Figura 49, en la cual se puede ver la jerarquía de cada uno de los tipos de objeto dentro de esta librería.

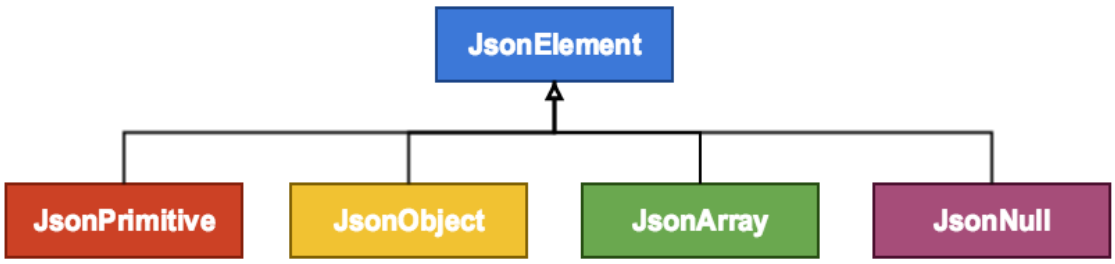


Figura 49 Diagrama de jerarquía de objetos en GSON

A su vez en la Figura 50 se puede visualizar de manera gráfica como se componen estos diferentes tipos de objetos, ejemplificándolo.

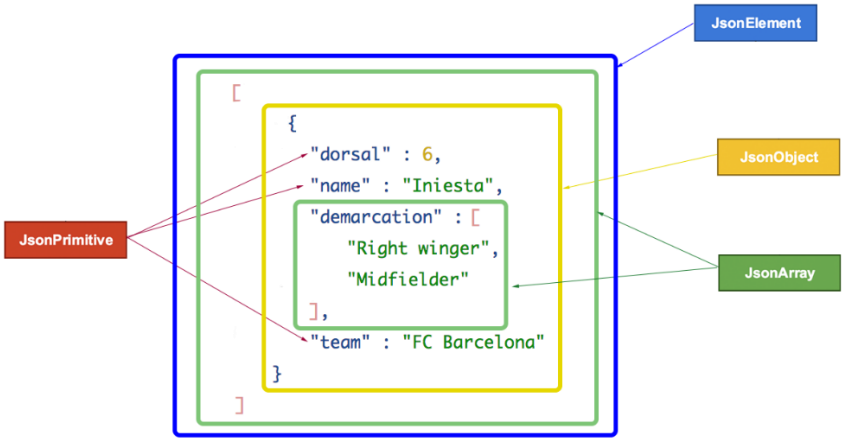


Figura 50 Ejemplificación de los objetos de GSON

Ejemplos prácticos de esta estructura son presentados más adelante en el desarrollo de la actividad.

Por definición todos los elementos del JSON son de tipo *JsonElement*, por lo que todo el JSON decimos que es un *JsonElement*. En primer lugar, nos encontramos con un *JsonArray* que es una lista (o array) de objetos. Esto lo podemos reconocer por el símbolo del corchete «[…]». Dentro de ese *JsonArray* no encontramos con un único objeto del tipo *JsonObject* y ese objeto a su vez contiene una seríe de *JsonElements* de los cuales 3 son *JsonPrimitives* y uno vuelve a ser un *JsonArray*.

La clase principal por usar es Gson, *que puede crear simplemente llamando a* new Gson (). *También hay una clase* GsonBuilder *disponible que se puede usar para crear una instancia de* Gson *con varias configuraciones como control de versiones, etc.*

La instancia de Gson *no mantiene ningún estado mientras invoca operaciones de* Json. *Por lo tanto, puede reutilizar el mismo objeto para múltiples operaciones de* serialización *y* deserialización *de* Json.

(Serialización)
Gson gson = new Gson ();
gson .toJson (1); ***==> imprime 1***
gson .toJson ("abcd"); ***==> imprime "abcd"***
gson .toJson (new Long (10)); ***==> imprime 10***
valores int [] = {1};
gson .toJson (valores); ***==> imprime [1]***
(Serialización)

Gson gson = new Gson ();
gson .toJson (1); ***==> imprime 1***
gson .toJson ("abcd"); ***==> imprime "abcd"***
gson .toJson (new Long (10)); ***==> imprime 10***
valores int [] = {1};
gson .toJson (valores); ***==> imprime [1]***

(Deserialización)
int uno = gson. fromJson ("1", int.class);
Entero uno = gson.fromJson ("1", Integer.class);
Long one = gson .fromJson ("1", Long.class);
Booleano falso = gson .fromJson ("falso", Booleano.
String str = gson .fromJson ("\" abc \ "" , String.class);
String anotherStr = gson .fromJson ("[\" abc \ "]" , String.class);

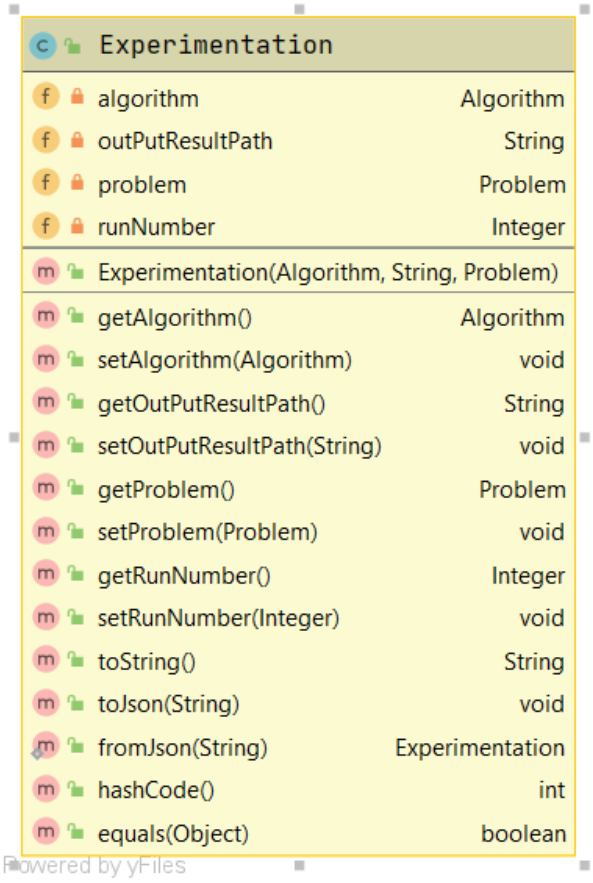
A.2 Definición de los objetos de experimento computacional de M-SDOSS para su uso en VisTHAA

En esta sección se presenta los códigos de los objetos y metidos definidos por el M-SDOSS y que VisTHAA usa a manera de guía a la hora de generar los objetos desde la configuración del experimento computacional.

Se presentan en el siguiente orden: tipo de problema, problema (junto con las cada uno de los problemas de los casos de estudio), algoritmo, operadores y finalmente solución, cabe destacar que todos los objetos siguen la estructura y comportamiento de un objeto del patrón conocido como constructor (Builder en inglés).

A.2.1 Experimento

La Figura 34 muestra la estructura los datos del problema, de la instancia y del objeto generado para el algoritmo. Con este objeto se permitirá la creación del archivo JSON, este objeto es la representación general del experimento computacional y todos los demás objetos están asociados a este de una u otra manera a excepción de solución.



A.2.2 Tipo de problema

La Figura 35 muestra la estructura los datos de los tipos de problema que son aceptados actualmente para realizar experimentos computacionales, Debido a que la definición de cada problema hereda de la clase problema aquí se agregan los métodos serializable y comparables dados por GSON para que los objetos puedan restructurarse de manera correcta al ser leído del archivo JSON. Con este objeto se permitirá la definición de problemas, este objeto es el esqueleto de los problemas que se trabajan.

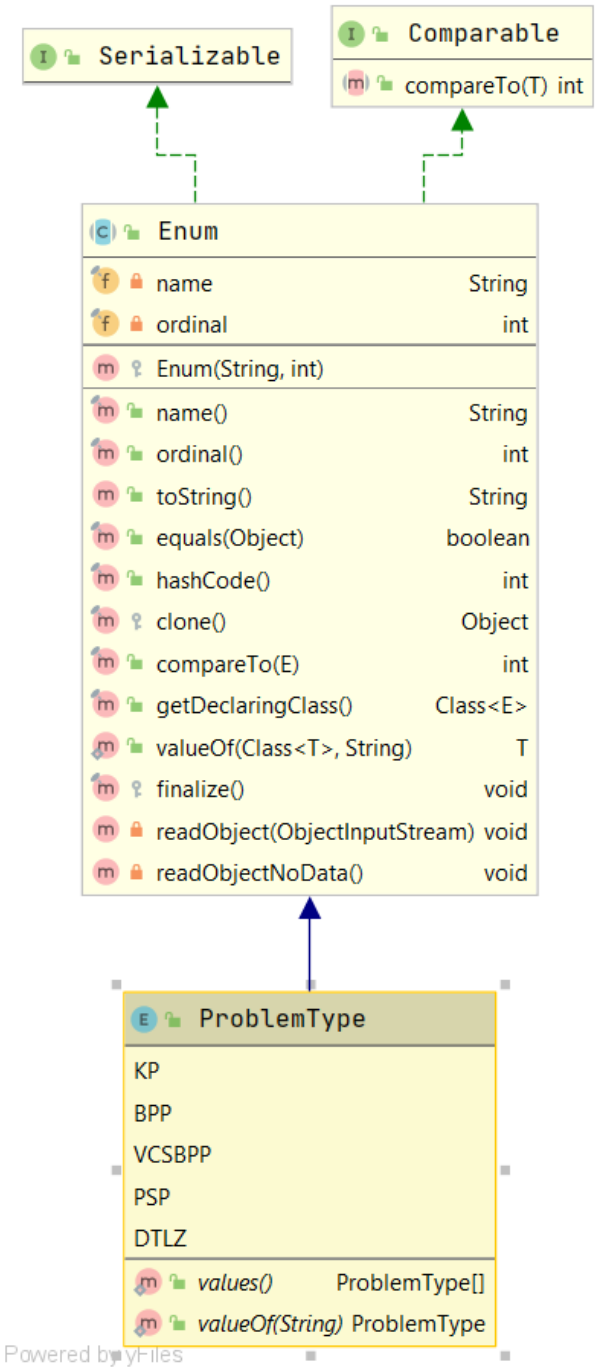


Figura 52 Diagrama de clase de tipo de problema

A.2.3 Problema

El objeto problem que se genera a partir del diagrama que se muestra en la Figura 36, es la descripción del objeto problem que está presente en experimento. El objeto algoritmo contienen los atributos generales de un problema, los problemas específicos heredan.

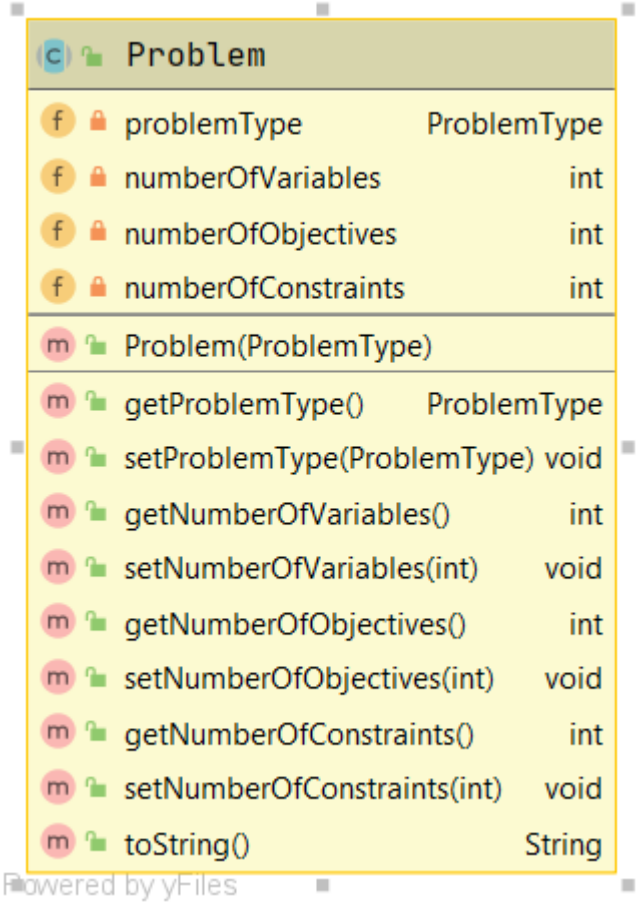


Figura 53 Diagrama de clase de problema

A.2.4 Knapsack 0/1

El objeto knapsack_problem que se genera a partir del diagrama que se muestra en la Figura 37, es la descripción del objeto knapsack_problem que está presente en experimento. El objeto contiene los atributos necesarios de su problema y solamente será parte del experimento computacional si es seleccionado explícitamente.

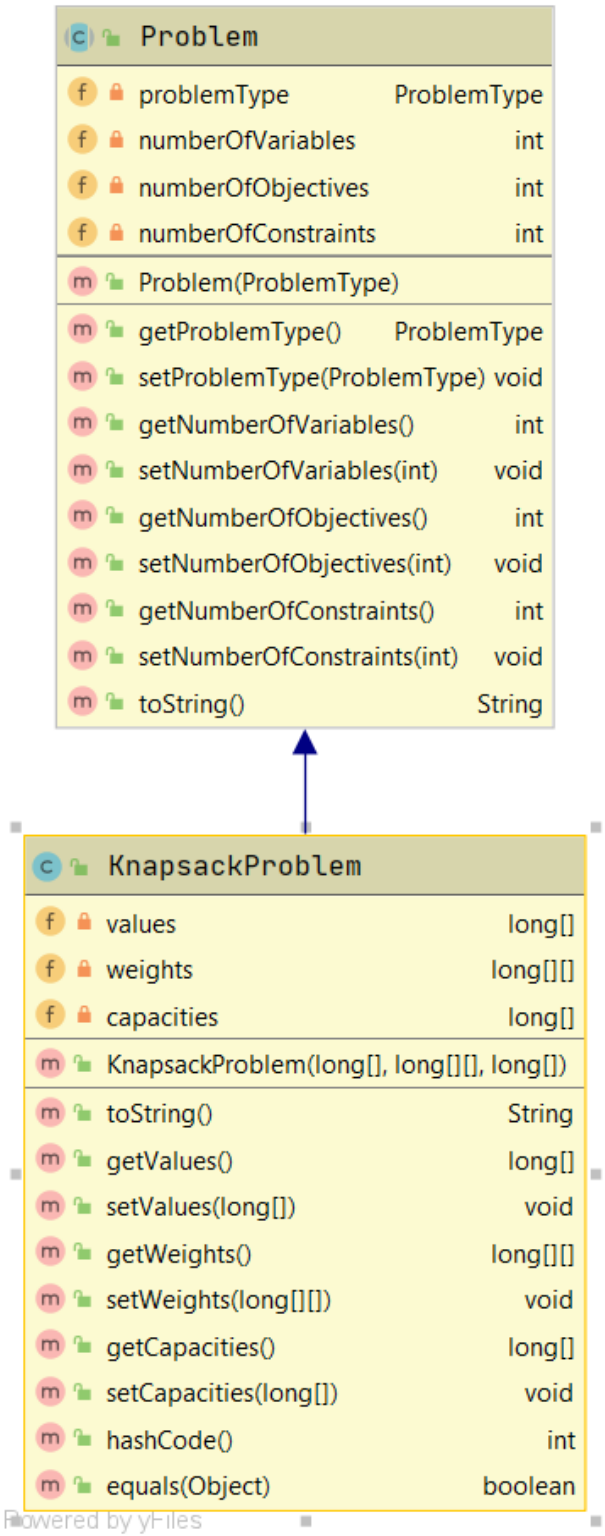


Figura 54 Diagrama de clase de Knapsack 0/1

A.2.5 PSP

El objeto psp que se genera a partir del diagrama que se muestra en la Figura 38, es la descripción del objeto psp que está presente en experimento. El objeto contiene los atributos necesarios de su problema y solamente será parte del experimento computacional si es seleccionado explícitamente.

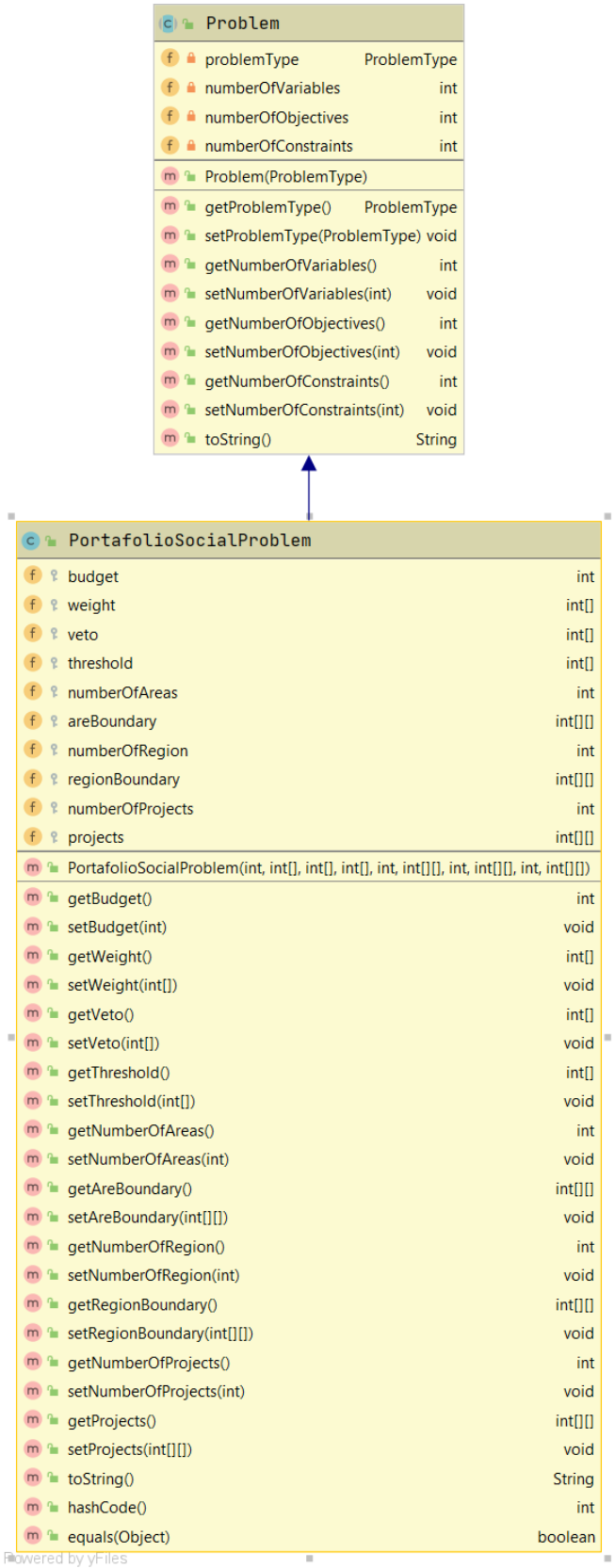


Figura 55 Diagrama de clase de PSP

A.2.6 DTLZ

El objeto DTLZ que se genera a partir del diagrama que se muestra en la Figura 39, es la descripción del objeto DTLZ que está presente en experimento. El objeto contiene los atributos necesarios de su problema y solamente será parte del experimento computacional si es seleccionado explícitamente, cabe resaltar que DTLZ genera un tipo de objeto basado en el DTLZ seleccionado dentro de VisTHAA (varia del 1 al 3).

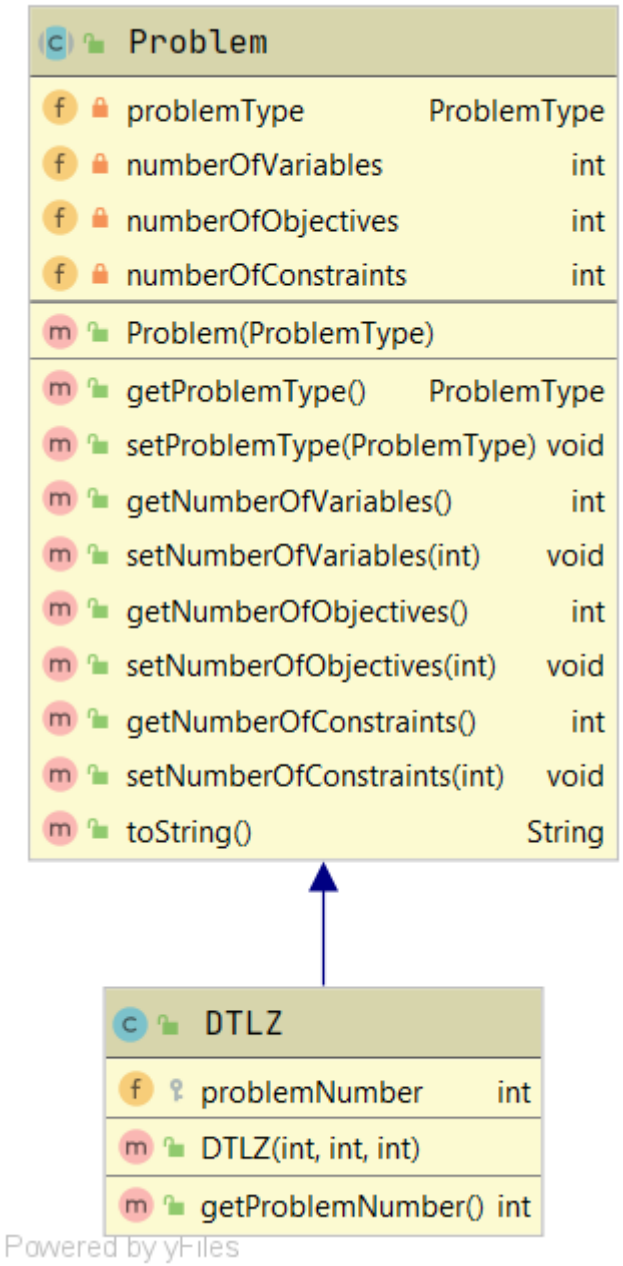


Figura 56 Diagrama de clase de DTLZ

A.2.7 Algoritmo

El objeto algoritmo que se genera a partir del diagrama que se muestra en la Figura 40, es la descripción del objeto algoritmo que está presente en experimento. El objeto algoritmo contienen los atributos necesarios de un algoritmo solucionador y los objetos generados para los operadores.

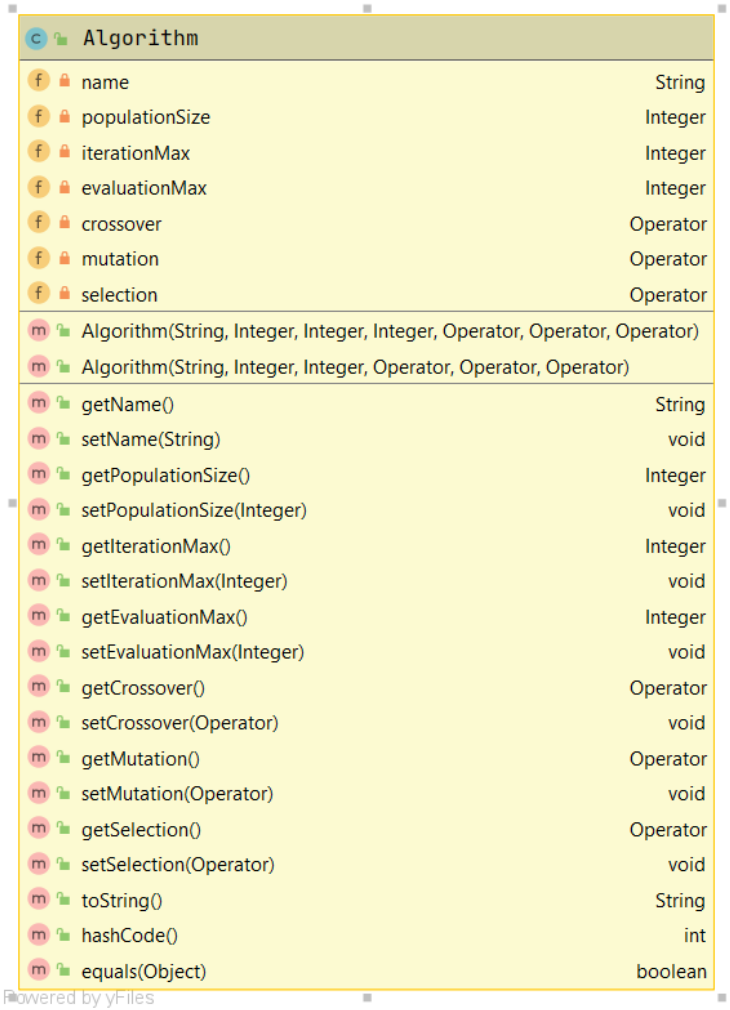


Figura 57 Diagrama de clase de algoritmo

A.2.8 Operadores

El objeto que se genera a partir del diagrama que se muestra en la Figura 41 del objeto operador, es la descripción del objeto operador que esta presenta en algoritmo. De este objeto se requiere generan tres, uno para selección, uno para la cruza y un para la muta. Contienen los atributos necesarios de un operador.

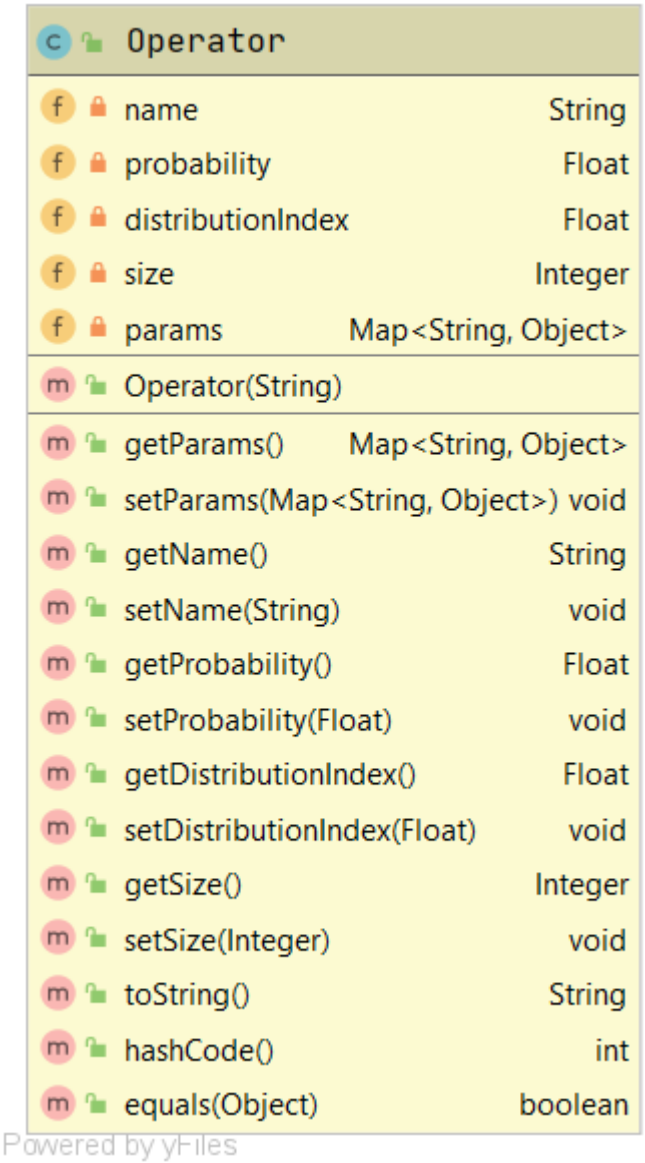


Figura 58 Diagrama de clase de operador

A.2.9 Solución

El objeto que se genera a partir del diagrama que se muestra en la Figura 42 del objeto solution, es la descripción del objeto solution que es la representación que se dará a la solución de M-SDOSS para que pueda ser interpretada por VisTHAA. De este objeto se requiere generan n objetos, siendo n el valor seleccionado para el número de ejecuciones del experimento computacional. Contienen los atributos necesarios de la solución.

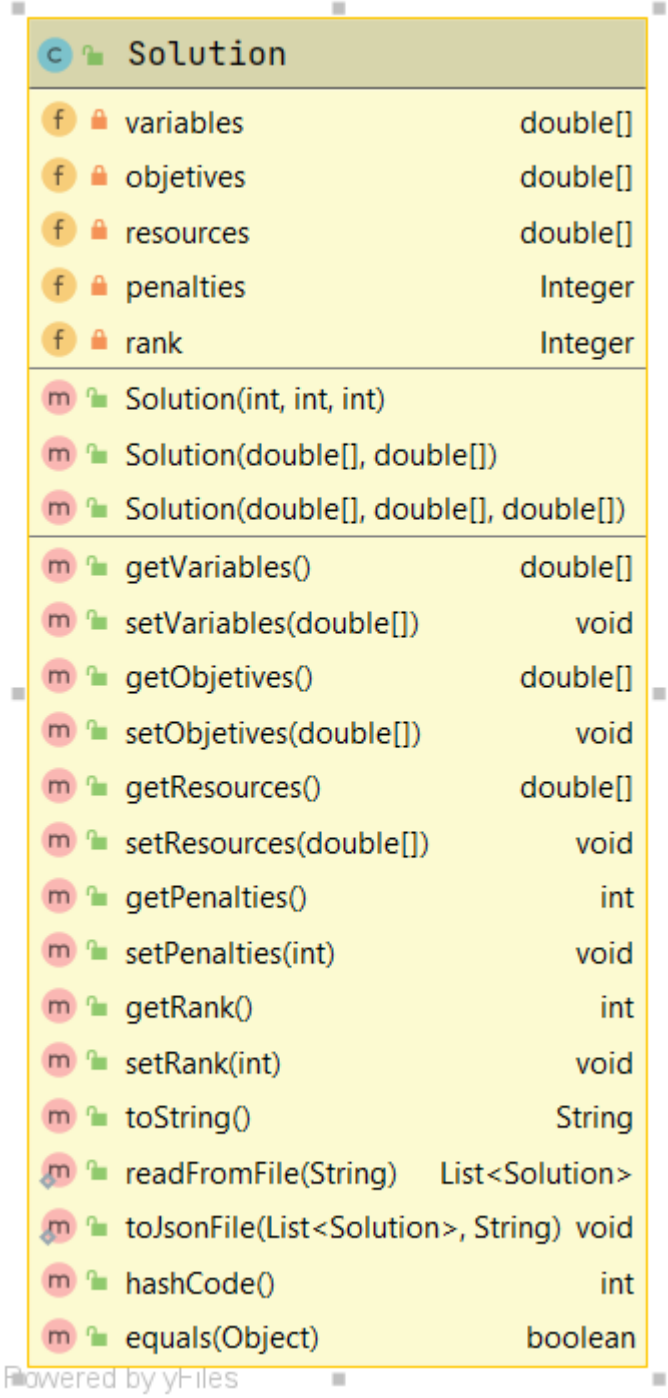


Figura 59 Diagrama de clase de solución

Anexo B

Análisis algoritmo genético

Dentro de este anexo se describirá las actividades realizada y los cambios dentro del proyecto de software VisTHAA en la sección del algoritmo genético se realizará una breve descripción de las funciones implementadas y el cambio realizado a la función callGenetic () ubicada en los scripts PerformanceCharacterizationProcessMonoObjetive y BehaviorCharacterizationProccess.

Dentro del proyecto de software VisTHAA se realizó el análisis y documentación sobre los siguientes paquetes, principalmente los métodos que responden al proceso de la ejecución del algoritmo genético:

- PerformanceCharacterization
  - PerformanceCharacterizationInterfaceMonoObjetive.java
  - PerformanceCharacterizationProcessMonoObjetive.java
- BehaviorCharacterization
  - BehaviorCharacterizationInterface.java
  - BehaviorCharacterizationProccess.java
- Algorithms
  - GeneticKP

Para la actividad en específico se realizar una proposición de un cambio dentro del método **callGenetic ()**, con la finalidad de mejorar el desempeño del algoritmo Genético de VisTHAA.

B.1 PerformanceCharacterization

Dentro de esta sección se realizar la descripción de la interfaz y de los métodos empleados que realiza dentro del proceso dentro de PerformanceCharacterization.

B.1.1 PerformanceCharacterizationInterfaceMonoObjetive.java

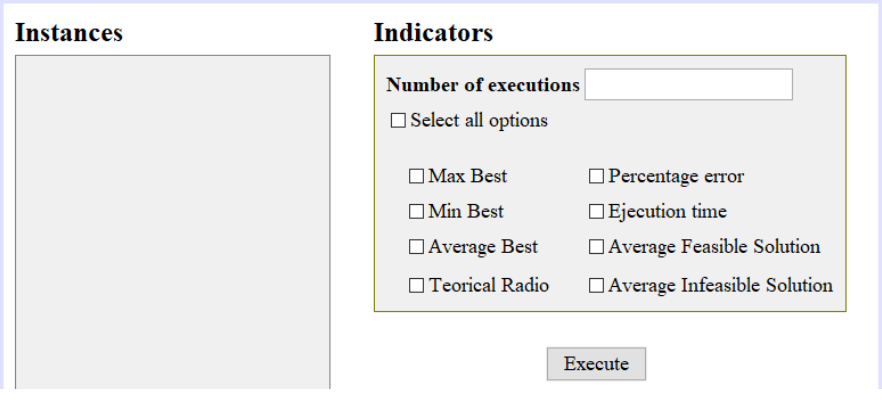


Figura 60 Interfaz de ventana PerformanceCharacterization

A continuación, se realizará la descripción de las opciones mostradas dentro de la anterior interfaz:

- **Number of executions:** Formulario donde se especifica el número de repeticiones que tendrá el algoritmo.
- **Select All options:** Booleano que marca todas las casillas de opciones.
- **Max Best:** Booleano que indica si se reportara el valor máximo encontrado en el archivo de texto generado.
- **Min Best:** Booleano que indica si se reportara el valor mínimo encontrado en el archivo de texto generado.
- **Average Best:**Boleano que indica se reportara el valor promedio de los mejores resultados en el archivo de texto generado.
- **Teorical Radio:** Booleano que indica si se reportara el porcentaje de cuan bueno es el resultado en base al valor teórico.
- **Percentage error:** Booleano que indica si se reportara el porcentaje de error con respecto al valor teórico.
- **Ejecution time:** Booleano que indica si se reportara el tiempo de duración de la ejecución.
- **Average Feasible Solution:** Booleano que indica si se realizara el reporte del valor promedio de soluciones factibles
- **Average Infeasible Solution:** Booleano que indica si se realizara el reporte del valor promedio de soluciones infactibles.
- **Execute:** Manda a llamar la función jButton1ActionPerformed el cual obtiene los índices de las opciones seleccionadas, realiza una verificación del valor de las iteraciones y luego procede a realizar la ejecución del genético verificando el tipo de problema es Bin Packing o Knapsack.

B.1.2 PerformanceCharacterizationProcessMonoObjetive.java

Dentro de esta subsección se realizará la descripción de los métodos empleados en el script de proceso.

- **countMeasurements:** Realiza un conteo de las métricas seleccionadas en PerformanceCharacterizationInterfaceMonoObjective.
- **callGenetic:** Realiza una verificación del logBookFile si es diferente de nulo realiza la lectura de la instancias en el logBookFile e inicializa el algoritmo genético para cada una de las instancias si no realiza el algoritmo genético para una única instancia una vez finaliza la ejecución realiza el guardado de los resultados en "%user.dir%\\VISTHAA\\Performance\\Performance Characterization\\" además muestra un mensaje en pantalla indicando la ruta donde se almaceno el archivo con los resultados.
- **getInputAsString:** esta función recibe un InputStream y retorna un string resultado de una delimitación del string con "\\A" en el InputStream si no existe retorna un string vacío.

- **CallVNSCSBPP:** Realiza la ejecución de un algoritmo de Búsqueda de vecindad variable para el problema de BinPaking una vez finaliza la ejecución realiza el guardado de los resultados en "%user.dir%\\VISTHAA\\Performance\\Performance Characterization\\" además muestra un mensaje en pantalla indicando la ruta donde se almaceno el archivo con los resultados.

B.2 BehaviorCharacterization

Dentro de esta sección se realizar la descripción de la interfaz y de los métodos empleados que realiza dentro del proceso de BehaviorCharacterization.

B.2.1 BehaviorCharacterizationInterface

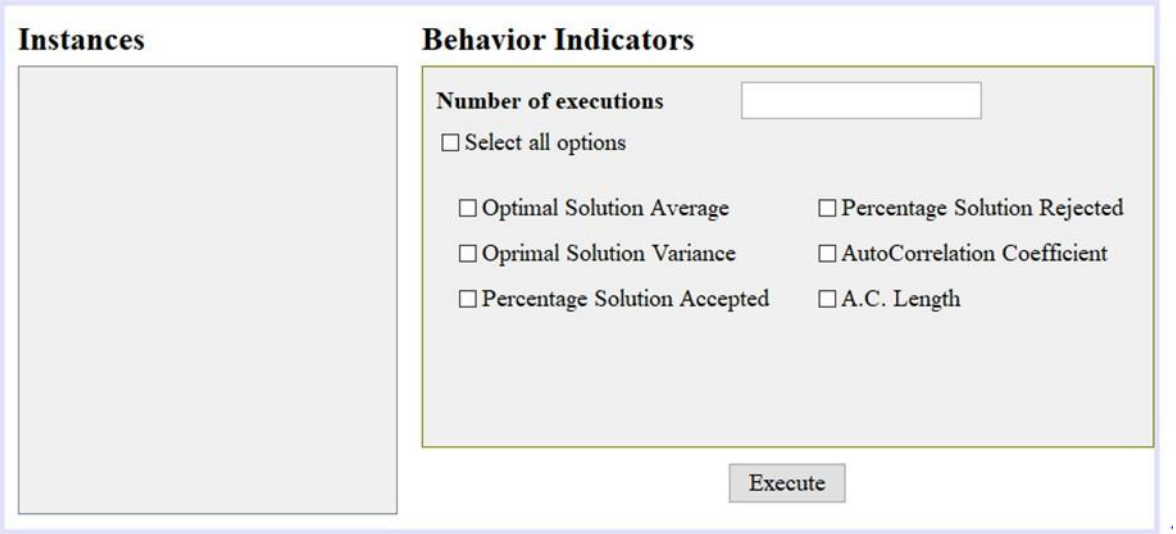


Figura 61 Interfaz de ventana BehaviorCharacterization

- **Number of executions:** Formulario donde se especifica el número de repeticiones que tendrá el algoritmo.
- **Select all options:** Booleano que marca todas las casillas de opciones.
- **Optimal Solution Average:** Booleano que indica si se reportara el valor optimo promedio encontrado en el archivo de texto generado.
- **Optimal Solution Variance**: Booleano que indica si se reportara el valor promedio de la variancia optima encontrada en el archivo de texto generado.
- **Percentage Solution Accepted:** Booleano que indica si se reportara el valor de porcentaje de solución aceptada.
- **AutoCorrelation Coefficient:** Booleano que indica si se reportara el valor del coeficiente de autocorrelación.
- **A.C. Length:** *

B.3 Algorithms

Dentro de esta sección se realizará la descripción de un algoritmo perteneciente al Sowftware de vista el cual es el GeneticKP.

B.3.1 GeneticKP

A continuación, se mostrarán los **parámetros** dentro de este script:

- **PoblacionGeneracional:** Matriz que contiene los Cromosoma de todos los individuos de la población.
- **poblacion:** Numero Entero que indica la cantidad de población.
- **generacion:** Numero de generaciones
- **porcentajeCruza:** Porcentaje de Cruza entre los individuos de cada generación
- **porcentajeMuta:** Porcentaje de Mutación de los individuos de cada generación
- **porcentajeBusquedaLocal:** Porcentaje de Búsqueda local Aplicado a los individuos.
- **indicesBarajeados:** vector que almacena los índices barajeados de la población.
- **aptitud:** Vector que indica la aptitud de cada individuo de la generación.
- **auxAptitud:** vector auxiliar donde se indican las aptitudes en base a los índices barajeados de los individuos.
- **matrizSeleccion:** matriz que contiene los índices de los padres.
- **elementos:** cantidad de objetos.
- **capacidad:** valor de peso total de la mochila
- **pesoBeneficio:** Matriz que contiene el peso y beneficio de cada uno de los objetos.
- **pesoBeneficio:** vector de beneficios de cada uno de los objetos
- **arrayPeso:** vector de pesos de cada uno de los objetos.
- **problema:** instancia de Problema Knapsack

A continuación, se realizará la descripción de los **métodos** aplicado en GeneticKP:

- **LecturaInstancias:** Realiza la lectura de una instancia si la variable logBookFile es diferente de nulo carga el problema llamando el método *loadProblem* si no asigna la dirección del archivo obteniendo la dirección absoluta de logbook. Luego de esto se lee el archivo y se asignan los parámetros: la cantidad de elementos, la capacidad, los pesos y beneficios.
- **PesoBeneficio:** Asigna los valores de la matriz pesoBeneficio a los respectivos vectores de arrayPeso y arrayBeneficio.
- **PoblacionFactible:** Genera una población Factible de manera aleatoria.

- **SeleccionTorneoDeterministico:** Realiza la selección de los padres empezando por el reset del vector de indicesBarajeados, realizar el cálculo de aptitud y aptitud auxiliar luego de esto llama la funcion barajeo de la población y luego realiza la asignación de la matriz de selección con respecto a los índices barajeados.
- **BarajarPoblacion:** Mediante un algoritmo aleatorio se realiza el barajeo intercambiando los índices de los individuos en el vector de IndicesBarajeados además de intercambiar las aptitudes en la variable de auxAptitud.
- **CruzaOrderCrossover:** realiza la cruza entre los padres haciendo uso de la matrizSeleccion
- **CruzaDosPuntos:** Toma un porcentaje de la población y luego realiza la cruza de dos puntos los hijos resultantes sustituyen a los padres en la población.
- **MutacionInsersion:** Toma un porcentaje de la población y por cada individuo selecciona una zona específica de genes haciendo un intercambio entre ellos.
- Mutacion2Flips: Selecciona un porcentaje de población en base al porcentaje de muta para realizar la selección de dos alelos de cada individuo y aplicarle un flip.
- **BusquedaLocal:** *
- **ElegidosRepetidos:** *
- **Peso:** Función que devuelve el peso de un individuo de la población mediante el índice.
- **Beneficio:** Función que devuelve el beneficio de un individuo de la población mediante el índice.
- **loadProblem:** Este método hace la lectura un archivo para la creación del tipo de clase KnapsackProblem.

B.4 Cambio realizado dentro de CallGentic

Dentro de la propuesta de modificación del método se dispuso la creación de dos clases para poder usar la paralelización de las experimentación e instancias y con esto reducir el tiempo de ejecución del algoritmo. Con la finalidad de implementar esto a continuación se explicará en subsecciones la documentación y ejemplos básicos de implementación.

B.4.1 Multithreading Java

Primeramente, se explicará lo que es un hilo o Thread que es básicamente un peso ligero y la unidad de procesamiento más pequeña. Java usa subprocesos utilizando una “clase de subprocesos”.

Cuando una aplicación comienza por primera vez, se crea el hilo de usuario. Publica eso, podemos crear muchos hilos de usuario e hilos daemon.

Multihtreading es el subprocesamiento múltiple en Java en el cual es un proceso de ejecución de dos o más subprocesos simultáneamente para una utilización máxima de la CPU.

Las aplicaciones multiproceso son donde dos o más hilos se ejecutan simultáneamente; por lo tanto, también se conoce como Concurrencia en Java. Esta multitarea está hecha, cuando múltiples procesos comparten recursos comunes como CPU, memoria, etc.

```java
demotest del paquete;
public class GuruThread1 implementa Runnable {
    / **
      * @param args
      * /
    public static void main(String[] args) {
        Enhebrar guruThread1 = new Thread("Guru1");
        Enhebrar guruThread2 = new Thread("Guru2");
        guruThread1.start();
        guruThread2.start();
        System.out.println("Los nombres de los hilos son los siguientes:");
        System.out.println(guruThread1.getName());
        System.out.println(guruThread2.getName());
    }
    @Anular    public void run() {   }
}
```

Figura 62 Ejemplo de hilo múltiple

Cada hilo se ejecuta en paralelo el uno al otro. Los subprocesos no asignan área de memoria separada; por lo tanto, ahorra memoria. Además, el cambio de contexto entre hilos lleva menos tiempo.

B.4.2 Descripción de cambios

Dentro de la clase GeneticKP utilizada dentro del método callGenetic, los métodos y funciones son de tipo estáticas, por lo cual no nos permite crear instancias de la clase para poder correr o ejecutar varios problemas en paralelo por lo cual se realizó una copia de dicha clase la cual se nombró como GeneticKP_1 dicha clase fue modificada de tal manera que se pudieran crear instancias del problema.

Como el método CallGenetic() se encuentra ubicada en PerformanceCharacterizationProcessMonoObjetive y BehaviorCharacterizationProccess se creó una interfaz para las respectivas interfaces ya que el método requiere los nombres de las instancias y el número de experimentaciones dicha interfaz es implementada en las interfaces para que cada script de proceso pueda inicializar los experimentos sin crear copias de la clase teniendo una única clase de GeneticKP_1.
Como para realizar el experimento se requiere del uso de la clase de IndicatorsGenetic se realizó la implementación de métodos no estáticos en la clase de manera que se puedan efectuar instancias para cada uno de los problemas.

Habiendo realizado estos cambios se realizó la clase ExperimentGenetic en esta clase se crea la instancia de un problema en donde ejecuta la resolución del problema. Por encima de esta clase esta InstanceGenetic la cual crea número de instancias de ExperimentGenetic en base al número de experimentos se requieran haciendo que se ejecuten de manera paralela.

B.4.3 Experimentación

B.4.3.1 Objetivo

El objetivo de la comparación de la ejecución del programa es evaluar el desempeño en cuanto a tiempo de los cambios realizados haciendo diferentes evaluaciones.

B.4.3.2 Descripción

Se probará las instancias encontradas dentro del proyecto las cuales son Instancia_001.KP e Instancia_002.KP usando diferentes números de experimentos para comprobar el tiempo de ejecución las cuales serán: 30,300,3,000 y 30,000.

B.4.3.3 Condiciones de Experimentación

Las pruebas fueron realizadas con las siguientes especificaciones de equipo de cómputo:

- Equipo con Procesador Six-Core AMD Ryzen 5 3600 con 32 GB de RAM
- NetBeans 12.2
- Sistema Operativo Windows 10
- Lenguaje de Programación Java

B.4.3.4 Resultados

Tabla 15 Y 7 Tiempo de ejecución previo al cambio y con el cambio

	30	300	3000	30000
Instancia_001.KP	0.243	0.597	5.702	56.91
Instancia_002.KP	0.72	0.582	5.635	56.355

	30	300	3000	30000
Instancia_001.KP	0.36	0.134	1.044	9.454
Instancia_002.KP	0.37	0.833	1.174	9.786

Como se puede observar el tiempo en de 3000 a 3000 tiene un tiempo de mejora significante. En este caso la variable que se modifica son el número de experimentaciones que en todo caso no es

de gran impacto al funcionamiento del algoritmo sin embargo si en vez de modificar el número de experimentaciones se modificar el número de generaciones que iterará cada experimento este al estar embebido en una clase que correrá de manera paralela tendrá in impacto similar a la hora de realizar las experimentaciones.

B.4.3.5 Observaciones

En base a los resultados obtenidos en tiempo de ejecución se puede verificar que el tiempo se ve reducido drásticamente para el numero de experimentaciones de 3,000 y 30,000 esto haciendo de uso de experimentación en paralelo haciendo uso de los hilos del CPU con un uso más exhaustivo del equipo de cómputo aprovechando las capacidades de Hardware utilizado.

Anexo C

Análisis caracterización multiobjetivo

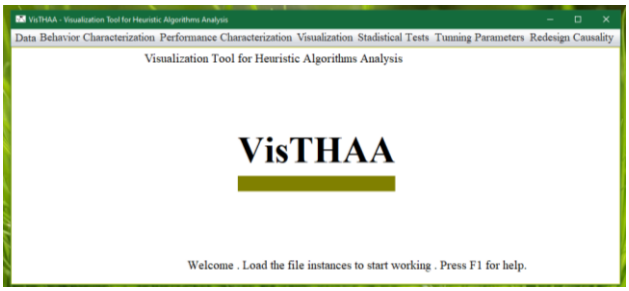
En este anexo se presentan el análisis al proceso de caracterización de desempeño y comportamiento de algoritmos multiobjetivo, así como al NSGA-II desarrollado dentro de VisTHAA. Además de algunas modificaciones realizadas dentro de los procesos.

C.1 PerformanceCharacterization

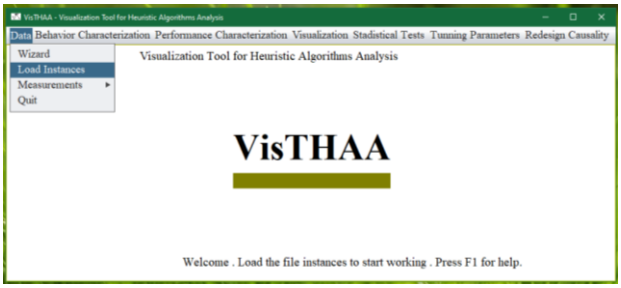
A continuación, se presenta el proceso para llevar a cabo la ejecución de la caracterización de desempeño multiobjetivo.

C.1.1 Ejecución de PerformanceCharacterization

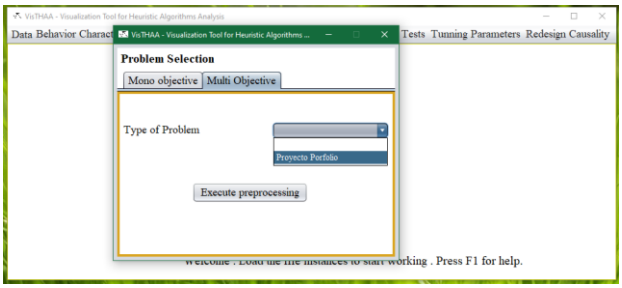
1.- Ejecutar VisTHAA



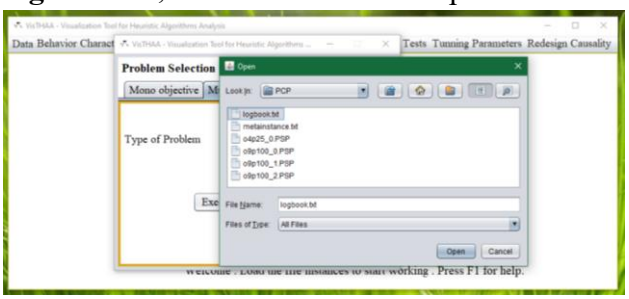
2.- En el menú data dar clic en **Load Instances**



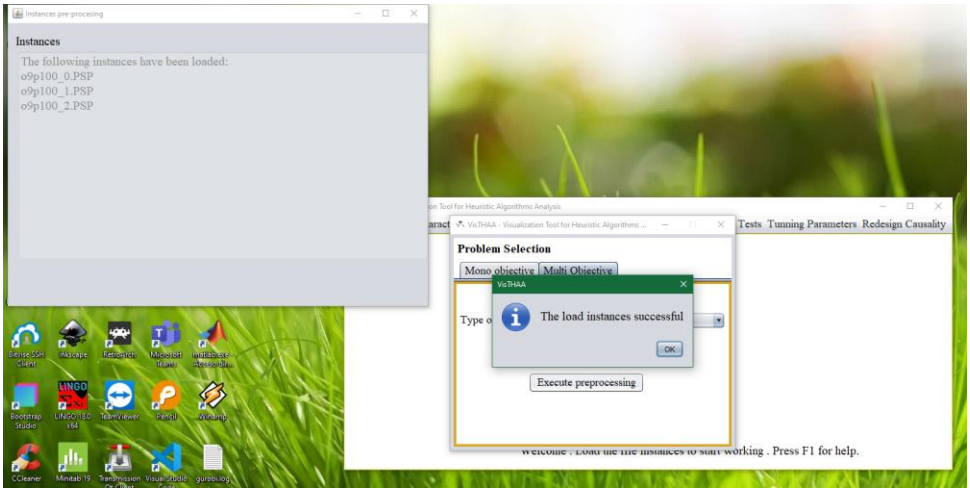
3.- Aparecerá un panel, dar clic en la pestaña **Multi Objective**, en **Type of Problem** seleccionar **Proyecto Porfolio** y dar clic en el botón **Execute preprocessing**



4.- En la carpeta donde tenga las instancias entrar a la carpeta **PCP** y seleccionar el archivo **logbook.txt**, dar clic en el botón Open



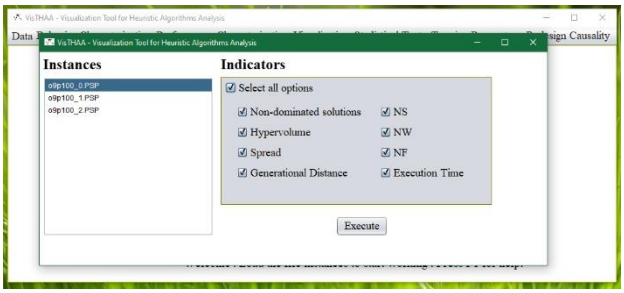
5.- Si las instancias son correctas, el programa nos indicará que las instancias se cargaron correctamente, de lo contrario nos lanzará un error, dar clic en el botón OK



6.- En la ventana principal de VisTHA damos clic en el menú **Performance Characterization** y damos clic en el submenú **Performance Multi Objetive**

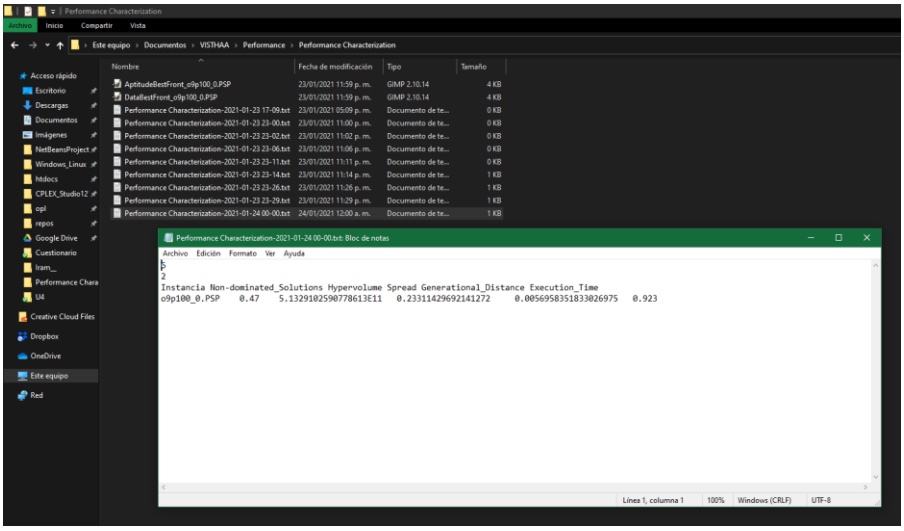

VisTHAA - Visualization Tool for Heuristic Algorithms Analysis
Data Behavior Characterization
Performance Characterization
Visualization
Stadistical Tests
Tunning Parameters
Redesign Causality
Performance Mono Objetive
Performance Multi Objetive
Algorithms Analysis
VisTHAA
Welcome . Load the file instances to start working . Press F1 for help.

7.- La aplicación nos mostrará un panel donde debemos seleccionar la instancia o instancias (se selecciona más de una presionando la tecla control y dando clic sobre la instancia) con las que queremos trabajar y palomear los indicadores que deseemos usar, una vez hecho eso, damos clic en el botón **Execute**



8.- Al terminar el proceso el sistema nos indicara que se guardaron los resultados en la carpeta VISTHA/Performance Characterization creada por el sistema en Documentos

9.- El archivo de resultados se guarda con el nombre Performance Characterization+ Fecha+Hora.txt



C.1.2 PerformanceCharacterizationInterfaceMultiObjetive.java

El archivo PerformanceCharacterizationInterfaceMultiObjetive.java tiene como objetivo mostrar al usuario un JFrame donde se pueda seleccionar una instancia o más instancias y seleccionar uno o varios indicadores de desempeño.

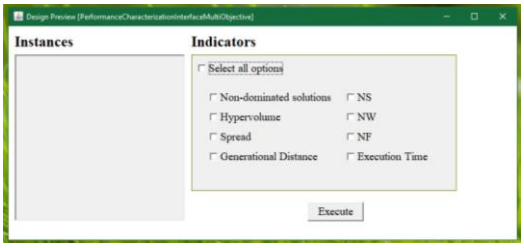


Figura 63 JFrame PerformanceCharacterizationInterfaceMultiObjetive

La selección que ha hecho el usuario de las instancias y los indicadores son enviados al objeto estático PerformanceCharacterizationProcessMultiObjetive al momento de dar clic en el botón Execute.

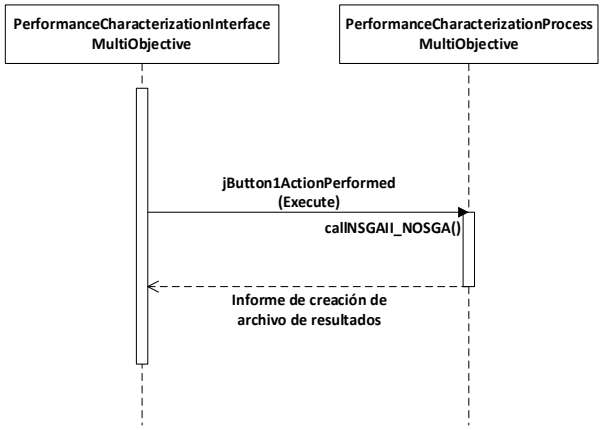


Figura 64 Comunicación entre GUI y Objeto performance characterization

Los indicadores seleccionados son almacenados en una vector de tipo entero llamado **seleccionados**, y las instancias en un ArrayList llamado **selected_instances_names**. El evento **jButton1ActionPerformed** ejecuta el método **callNSGAII_NOSGA()** de la clase PerformanceCharacterizationProcessMultiObjetive.
El diagrama UML de las clases PerformanceCharacterizationProcessMultiObjetive y PerformanceCharacterizationInterfaceMultiObjetive se muestra en la Figura 65.

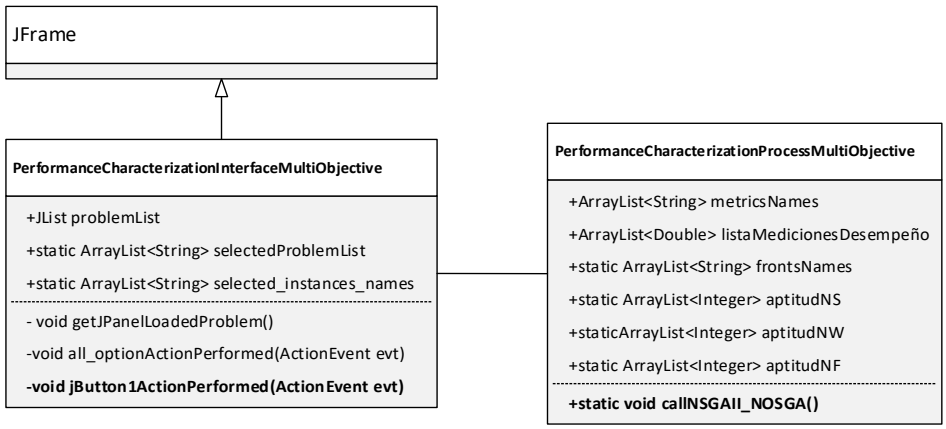


Figura 65 Diagrama UML de clases performance characterization

C.1.3 PerformanceCharacterizationProcessMultiObjetive.java

El evento **jButton1ActionPerformed** ejecuta el método **callNSGAII_NOSGA()** de la clase PerformanceCharacterizationProcessMultiObjetive, el cual desencadena el siguiente proceso:

1. INICIO
2. SI proportion_non_dominated esta seleccionado agregar a metricsNames "Non-dominated_Solutions"
3. SI hypervolume esta seleccionado agregar a metricsNames "Non-Hypervolume"
4. SI spread esta seleccionado agregar a metricsNames "Spread"
5. SI generational_distance esta seleccionado agregar a metricsNames "Generational_Distance"
6. SI executionTime esta seleccionado agregar a metricsNames "Execution_Time"
7. SI ns esta seleccionado agregar a frontsNames "NS"
8. SI nw esta seleccionado agregar a frontsNames "NW"
9. SI nf esta seleccionado agregar a frontsNames "NF"
10. FOR i=0 hasta numeroDeInstanciasSeleccionadas -1
    a. Leer nombre de instancia i
    b. Abrir instancia leida con objeto Algorithms.NSGAIIPCP
    c. Generar población inicial con NSGAIIPCP
    d. Calcular Aptitud de la población con NSGAIIPCP
    e. Seleccion por torneo con NSGAIIPCP
    f. Hacer generaciones = 0
    g. MIENTRAS generaciones < 1000
    h. Cruza con OrderCrossover con NSGAIIPCP
    i. Muta con MutacionInsercion con NSGAIIPCP
    j. generaciones = generaciones + 1
    k. FIN MIENTRAS
    l. Calcular FastDominating con NSGAIIPCP
    m. Calcular ElementosFrenteCero con Objeto Topsis
    n. Con PreprocesamientoNSGAII calcular:
        i. MatrizAptitudFrenteCero

ii. CalculoAptitudMejorFrente
iii. GuardadoAptitudMejorFrente en instancia leida

o. Hacer frenteCero = LecturaAptitud desde instancia leida
p. Calcular DistanciaEuclidiana con PerformanceNSGAII
q. Para PerformanceCharacterizationInterfaceMultiObjective:
r. SI proportion_non_dominated esta seleccionado agregar a listaMedicionesDesempeño
s. SI hypervolume esta seleccionado agregar a listaMedicionesDesempeño
t. SI spread esta seleccionado agregar a listaMedicionesDesempeño
u. SI generational_distance esta seleccionado agregar a listaMedicionesDesempeño
v. SI executionTime esta seleccionado agregar a listaMedicionesDesempeño
w. Guarda Informacion de NOSGA para cada frente seleccionado en archivo "Performance Characterization NOSGA-YYYY-MM-DD HH-MM.txt"

11. FIN FOR
12. Guarda Informacion de PerformanceCharacterizationProcessMultiObjective en el archivo Performance Characterization-YYYY-MM-DD HH-MM.txt
13. FIN

El proceso anterior se muestra en la Figura 66, en un diagrama de flujo de funciones cruzadas:

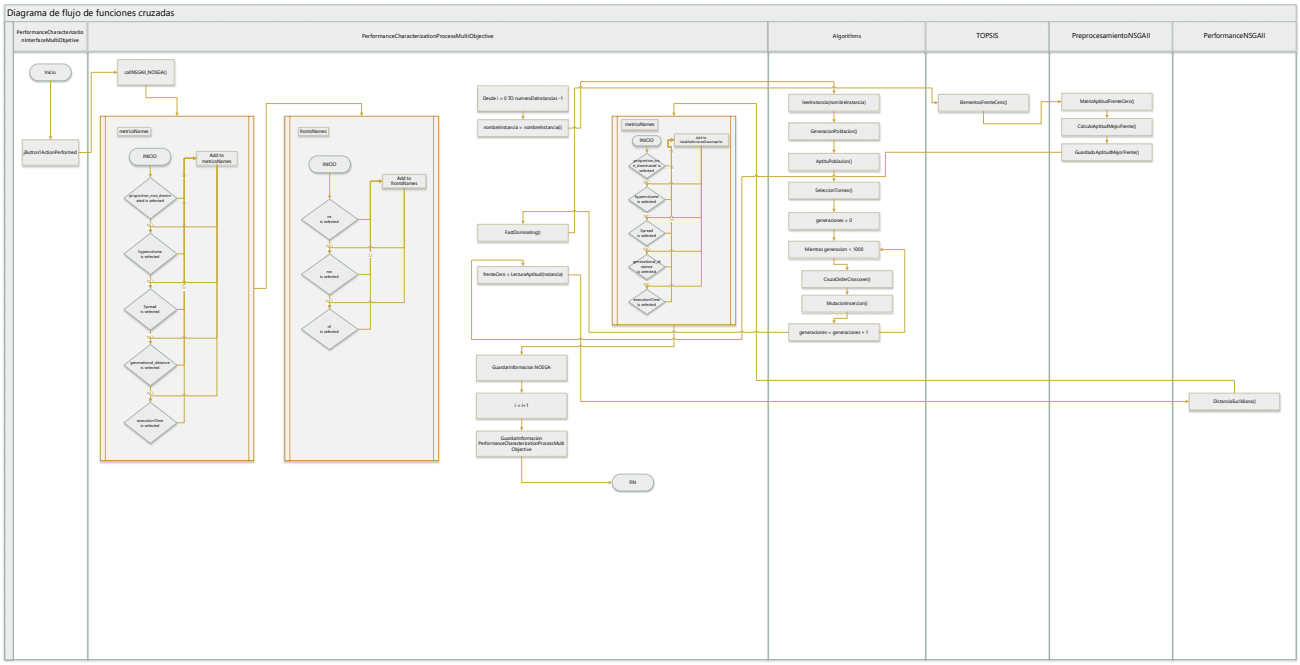


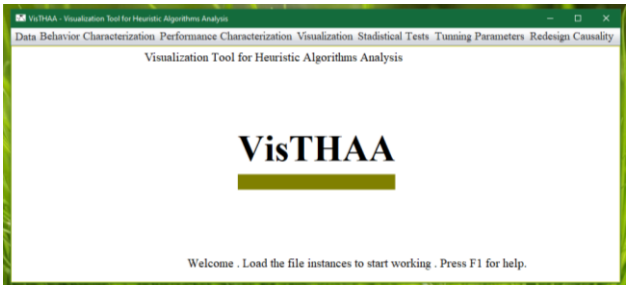
Figura 66 Diagrama de flujo de funciones cruzadas

C.2 BehaviorCharacterization

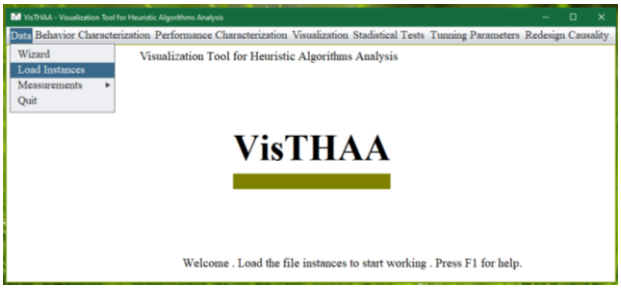
A continuación, se presenta el proceso para llevar a cabo la ejecución de la caracterización de comportamiento multiobjetivo.

C.2.1 Ejecución de BehaviorCharacterization

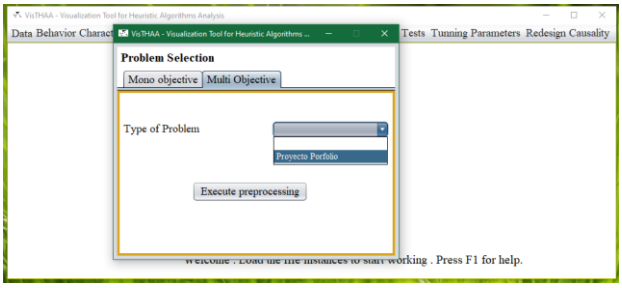
1.- Ejecutar VisTHAA



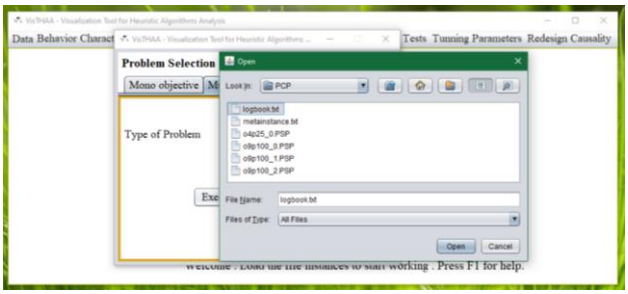
2.- En el menú data dar clic en **Load Instances**



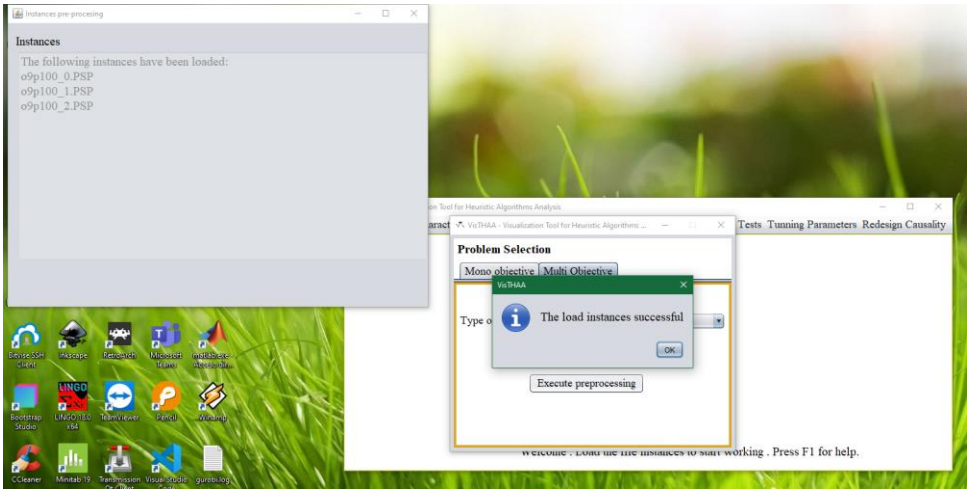
3.- Aparecerá un panel, dar clic en la pestaña **Multi Objective**, en **Type of Problem** seleccionar **Proyecto Porfolio** y dar clic en el botón **Execute preprocessing**



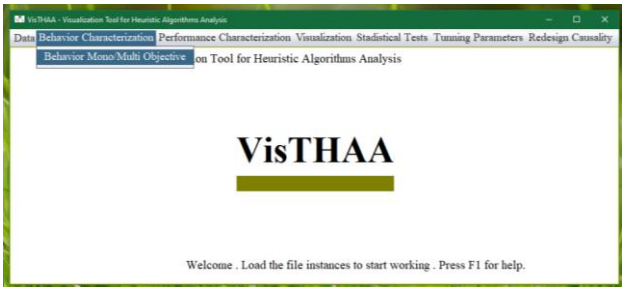
4.- En la carpeta donde tenga las instancias entrar a la carpeta **PCP** y seleccionar el archivo **logbook.txt**, dar clic en el botón Open



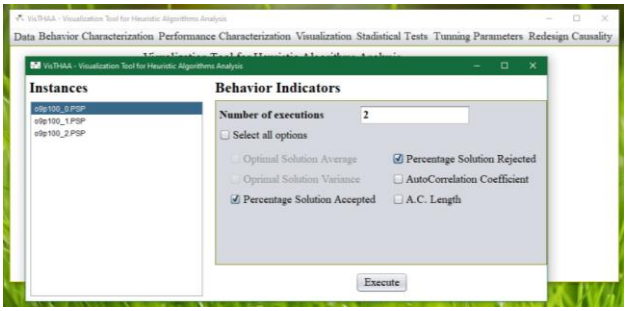
5.- Si las instancias son correctas, el programa nos indicará que las instancias se cargaron correctamente, de lo contrario nos lanzará un error, dar clic en el botón OK



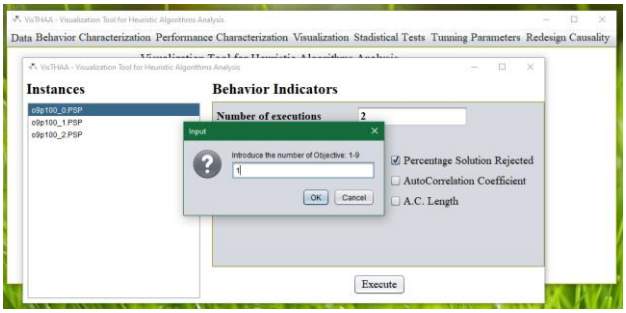
6.- En la ventana principal de VisTHA damos clic en el menú **BehaviorCharacterization** y damos clic en el submenú **Behavior Mono/Multi Objetive**



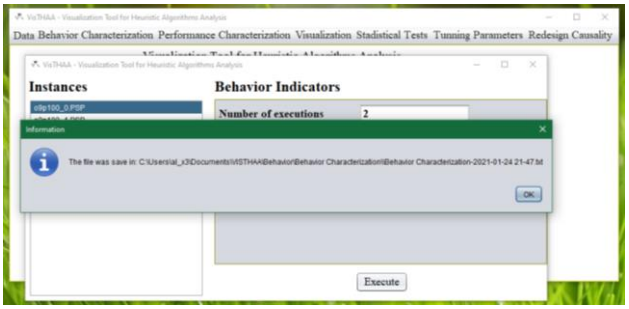
7.- Seleccionamos las instancias, definimos el número de ejecuciones y seleccionamos las opciones deseadas, damos clic en el botón **Execute**

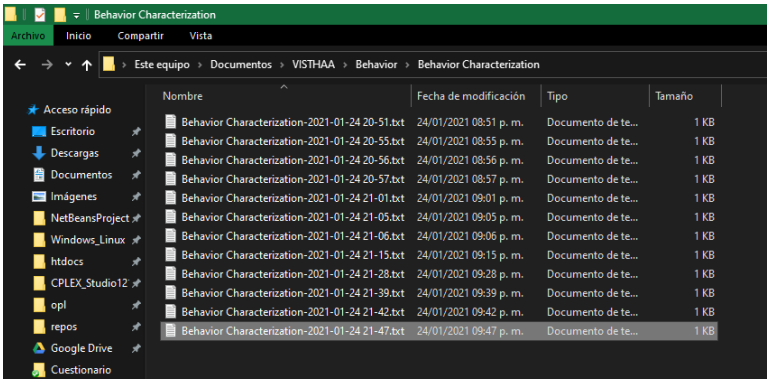


8.- Nos pedirá escribir el número del objetivo del 1 al 9, esto lo pedirá tantas veces como el número de ejecuciones, anotar el número y dar clic en el botón **OK**



9.- Al finalizar el sistema nos indicará la ruta y el nombre del archivo con los resultados





C.2.2 BehaviorCharacterizationInterface.java

La clase BehaviorCharacterizationInterface hereda de un JFrame, el cual permite hacer la selección de las instancias con las que queremos trabajar, asi como el número de ejecuciones y las opciones como Optmimal solution average, Optimal solution variance, Percentage Solution Accepted, Percentage Solution Rejected, AutoCorrelation Coefficient, y A. C. Length.

Dependiendo si trabajamos con Mono Objetivo o Multi Objetivo al presionar el botón Execute se pueden lanzar los métodos callGenetic(), callVNSVCSBPP() o callNSGAII().

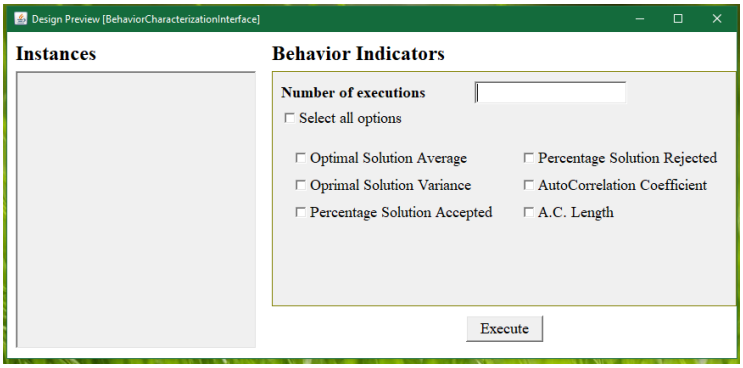


Figura 67 JFrame BehaviorCharacterizationInterface

El método callGenetic() corresponde al problema de Knapsack Mono Objetivo, callVNSVCSBPP() corresponde al problema Bin Packing Mono Objetivo, callNSGAII() corresponde al problema Porfolio Multi Objetivo.

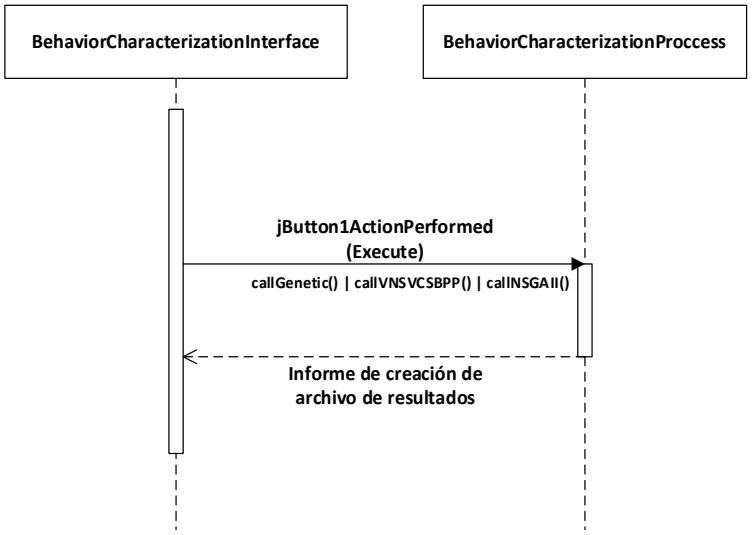


Figura 68 Comunicación entre GUI y Objeto Behavior characterization

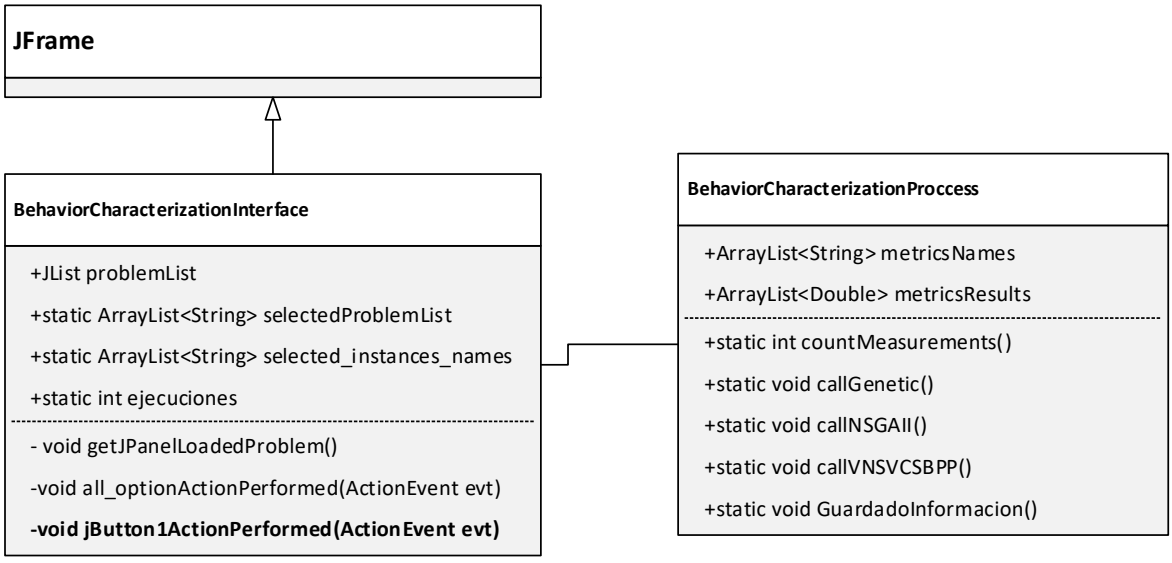


Figura 69 Diagrama UML de clases Behavior characterization

C.2.3 BehaviorCharacterizationProccess.java

A continuación, se enumera para cada método de la clase BehaviorCharacterizationProccess el proceso de operación:

Método countMeasurements

1. Hacer selected_measurements_names = 0;
2. Si esta seleccionada optimalSolutionAverage agregar "Optimal_Solution_Average" a metricsNames
3. Hacer selected_measurements_names = selected_measurements_names + 1
4. Si esta seleccionada optimalSolutionVariance agregar "Optimal_Solution_Variance" a metricsNames
5. Hacer selected_measurements_names = selected_measurements_names + 1
6. Si esta seleccionada percentageSolutionAccepted agregar "Percentage_Solution_Accepted" a metricsNames
7. Hacer selected_measurements_names = selected_measurements_names + 1
8. Si esta seleccionada percentageSolutionRejected agregar "Percentage_Solution_Rejected" a metricsNames
9. Hacer selected_measurements_names = selected_measurements_names + 1
10. Si esta seleccionada autoCorrelationCoefficient agregar "AutoCorrelation_Coefficient" a metricsNames
11. Hacer selected_measurements_names = selected_measurements_names + 1
12. Si esta seleccionada autoCorrelationCoefficientLength agregar "AutoCorrelation_Coefficient_Length" a metricsNames
13. Hacer selected_measurements_names = selected_measurements_names + 1

Método callGenetic

1. Llamar metodo countMeasurements
2. FOR i=0 hasta numeroInstanciasSeleccionadas - 1
3. Hacer nombreInstancia = intancia i
4. Desde Algorithms.GeneticKP
    i. Leer Instancia nombreInstancia
    ii. generar PoblacionFactible
    iii. ejecutar SeleccionTorneoDeterministico
5. double datos[numeroEjecuciones]
6. FOR j=0 hasta numeroEjecuciones - 1
    i. Hacer generaciones = 0
    ii. WHILE generaciones < Algorithms.GeneticKP.generacion
        1. Desde Algorithms.GeneticKP
            a. Ejecutar CruzaOrderCrossover
            b. Ejecutar MutacionInsersion
    iii. END WHILE
    iv. Desde Algorithms.GeneticKP
        1. Ejecutar BusquedaLocal
    v. Hacer datos[j] = IndicatorsGenetic.MaxBest()
7. FIN FOR
8. Si esta seleccionado optimalSolutionAverage agregar a metricsResults "optimalSolutionAverage"
9. Si esta seleccionado percentageSolutionAccepted agregar a metricsResults "percentageSolutionAccepted"
10. Si esta seleccionado percentageSolutionRejected agregar a metricsResults "percentageSolutionRejected"
11. Si esta seleccionado autoCorrelationCoefficient agregar a metricsResults "autoCorrelationCoefficient"
12. Si esta seleccionado autoCorrelationCoefficientLength agregar a metricsResults "autoCorrelationCoefficientLength"
13. Si esta seleccionado autoCorrelationCoefficientLength agregar a metricsResults "autoCorrelationCoefficientLength"
14. FIN FOR
15. Ejecuta metodo GuardadoInformacion()

Método callNSGAII

1. Llamar metodo countMeasurements
2. FOR i=0 hasta numeroInstanciasSeleccionadas - 1
3. Hacer nombreInstancia = intancia i
4. Desde Algorithms.NSGAIIPCP
    i. Leer Instancia nombreInstancia
    ii. generar PoblacionFactible
    iii. ejecutar SeleccionTorneoDeterministico
5. double datos[numeroEjecuciones]
6. FOR j=0 hasta numeroEjecuciones - 1
    i. Hacer generaciones = 0
    ii. WHILE generaciones < 1000
        1. Desde Algorithms.NSGAIIPCP
            a. Ejecutar CruzaOrderCrossover
            b. Ejecutar MutacionInsersion
    iii. END WHILE
    iv. Desde Algorithms.NSGAIIPCP
        1. Ejecutar FastDominating
        2. Ejecutar ContadorF0
    v. Desde Topsis
        1. Ejecutar MatrizAptitudF0
        2. Ejecutar Topsis
    vi. Leer objetivos
    vii. Hacer datos[j] = matrizAptitudFrenteCero
7. FIN FOR
8. Si esta seleccionado optimalSolutionAverage agregar a metricsResults "optimalSolutionAverage"
9. Si esta seleccionado percentageSolutionAccepted agregar a metricsResults "percentageSolutionAccepted"
10. Si esta seleccionado percentageSolutionRejected agregar a metricsResults "percentageSolutionRejected"
11. Si esta seleccionado autoCorrelationCoefficient agregar a metricsResults "autoCorrelationCoefficient"
12. Si esta seleccionado autoCorrelationCoefficientLength agregar a metricsResults "autoCorrelationCoefficientLength"
13. FIN FOR
14. Ejecuta metodo GuardadoInformacion()

Método callVNSVCSBPP

1. Llamar metodo countMeasurements
2. FOR i=0 hasta numeroInstanciasSeleccionadas - 1
3. Hacer nombreInstancia = intancia i
4. Desde Algorithms.VNSVCSBPP
    i. Leer Instancia nombreInstancia
5. double datos[numeroEjecuciones]
6. FOR j=0 hasta numeroEjecuciones - 1
    i. Hacer datos[j] = VNSVCSBPP.solant
7. FIN FOR
8. Si esta seleccionado optimalSolutionAverage agregar a metricsResults "optimalSolutionAverage"
9. Si esta seleccionado optimalSolutionVariance agregar a metricsResults "optimalSolutionVariance"
10. FIN FOR
11. Ejecuta metodo GuardadoInformacion()

C.3 Archivos Modificados

Se modificaron los siguientes archivos:

Archivo	**PerformanceCharacterizationProcessMultiObjective.java**
Ruta	src/PerformanceCharacterization/
Comentario	Se corrigieron rutas estáticas, se agregó PATH de usuario dinámico

Archivo	**BehaviorCharacterizationProccess.java**
Ruta	src/BehaviorCharacterization/
Comentario	Se añadió mensaje de archivo creado con ruta, anteriormente daba la sensación de que el modulo BehaviorCharacterization no hacia nada.