Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Interfaz gráfica para el análisis de meta-heurísticas y para la visualización de soluciones a problemas de optimización combinatoria

Graphic interface for metaheuristic analysis and visualization of optimization combinatorial problems solution

Carolina Candelaria Álvarez Martín

La Laguna, 26 de mayo de 2023

D. **Gara Miranda Valladares**, con N.I.F. 78.563.584-T profesora Titular de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **Eduardo Manuel Segredo González**, con N.I.F. 78.564.242-Z profesor Ayudante Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Interfaz gráfica para el análisis de meta-heurísticas y para la visualización de soluciones a problemas de optimización combinatoria*"

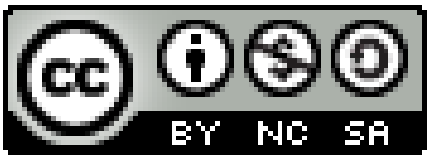
ha sido realizada bajo su dirección por D. **Carolina Candelaria Álvarez Martín**, con N.I.F. 46.244.488-K.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 26 de mayo de 2023

Agradecimientos

A mis padres, por nunca darse por vencidos.
A mis tutores, Gara Miranda Valladares y Eduardo Manuel Segredo González por su paciencia y la transmisión de sus conocimientos.
A mi pareja, por siempre estar ahí para mí.
A mi amigo Enrique, por recordarme las cosas realmente importantes.

Licencia



Resumen

Los problemas de optimización complejos podemos reconocer que suponen un reto en el momento de encontrar una solución óptima o una solución cercana a la óptima en un tiempo computacionalmente viable. Para ello se hacen uso de métodos de optimización mediante metaheurísticas que ofrecen soluciones factibles en un tiempo razonable para este tipo de problemas. Aunque es necesario tanto tener un conocimiento profundo del problema a tratar como de la técnica algorítmica que se va a utilizar. Las metaheurísticas bioinspiradas hacen uso de técnicas basadas en la naturaleza tales como la evolución biológica, la selección natural y la mutación, a lo largo de las diferentes evaluaciones. En cada evaluación se seleccionan aquellas soluciones que maximicen o minimicen la función objetivo para seguir evolucionando .

A lo largo de los años han nacido herramientas que facilitan el uso de metaheurísticas separando la definición del problema de la implementación del resolutor en sí mismo.Permitiendo utilizarlas centrándose en realizar una buena definición del problema en lugar de en las técnicas algorítmicas a utilizar. Este proyecto nace de la necesidad de implementar una interfaz gráfica que mejore y facilite la observación de las soluciones generadas en los estudios de viabilidad de una o varias técnicas metaheurísticas aplicadas a problemas de optimización.

Haciendo uso de librerías de python tales como Bokeh o Pandas se ha conseguido el desarrollo de una aplicativo web en el que se ofrece una serie de herramientas que permiten la evaluación de un conjunto de soluciones generadas por un algoritmo de optimización mediante un análisis estadístico y una visualización gráfica de las mismas. Empleando un fichero en formato JSON para la introducción de los datos del estudio haciendo que sea posible la automatización de la entrada de las soluciones para su posterior visualización y generación de estadísticas. El aplicativo también ofrece una funcionalidad para observar la evolución de la calidad de las soluciones a lo largo de las evaluaciones mediante una animación, brindando una perspectiva dinámica de cómo los algoritmos progresan y evolucionan a lo largo del tiempo.

En conclusión, se ha implementado una herramienta poderosa para observar y mejorar la calidad de las soluciones a problemas de optimización complejos y que podría mejorarse en un futuro para la visualización de las soluciones en la dimensión de las variables.

Palabras clave: meta-heurísticas, interfaz gráfica,problemas de optimización, gráficas, estadísticas, herramientas de resolución.

Abstract

Bioinspired metaheuristics make use of nature-based techniques, such as biological evolution, natural selection, and mutation, throughout the different evaluations. In each evaluation, solutions that maximize or minimize the objective function are selected to continue evolving.

Tools have emerged that facilitate the use of metaheuristics by separating the problem definition from the implementation of the solver itself. This allows users to focus on defining the problem effectively rather than on the specific algorithmic techniques to be used. This project arises from the need to implement a graphical user interface that enhances and simplifies the observation of solutions generated in feasibility studies of one or multiple metaheuristic techniques applied to optimization problems.

By utilizing Python libraries such as Bokeh and Pandas, a web application has been developed that provides a set of tools for evaluating a set of solutions generated by an optimization algorithm through statistical analysis and graphical visualization. This is achieved by employing a JSON file format for inputting study data, enabling the automation of solution input for subsequent visualization and statistical generation. The application also includes functionality for observing the quality evolution of solutions throughout evaluations through animation, offering a dynamic perspective on how algorithms progress and evolve over time. With the aid of this application, users can gain insights into the performance and effectiveness of optimization algorithms by visually analyzing the generated solutions and their statistical characteristics

In conclusion, a powerful tool has been implemented to observe and enhance the quality of solutions to complex optimization problems. This tool can be further improved in the future to enable visualization of solutions in the variable dimension. It provides users with the ability to analyze and evaluate the effectiveness of optimization algorithms through statistical analysis, graphical visualization, and dynamic animation.

Keywords: metaheuristics, graphic interface, optimization problems, plots, dataframes, statistics, solving tools.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Dentro del campo de la optimización el objetivo principal de esta es encontrar la solución óptima a un problema dentro de un conjunto de soluciones factibles, con un coste computacional razonable. Los modelos de optimización constan de tres componentes básicos:

- **Variables de decisión:** Representan los valores que se pueden tomar para modificar el valor de la función objetivo. Un conjunto de variables de decisión se corresponde con una solución concreta para el problema.
- **Restricciones:** Representan el conjunto de relaciones que las variables de decisión deben satisfacer para hacer una solución válida para ese problema.
- **Función objetivo:** Representa la función de evaluación que se usará para medir la calidad de las soluciones.

Para encontrar un método de resolución para un problema de optimización hay que tener en cuenta las necesidades de este. No es recomendable la utilización de un método exacto ya que aunque encuentran la solución óptima sólo son viables para problemas o instancias sencillos. Las heurísticas manejan conocimientos específicos de los problemas y obtienen soluciones de calidad sin un tiempo computacional excesivo.

Con el éxito de las metaheurísticas en entornos de optimización reales y complejos han surgido paradigmas de metaheurísticas bioinspiradas, basados en la biología tales como selección natural, colonia de hormigas, algoritmos genéticos.Estos paradigmas parecen adecuados para quienes estén interesados en obtener una solución buena para el problema sin invertir mucho tiempo en la comprensión matemática del modelo ni en el diseño a medida de algoritmos. Sin embargo, puede surgir la duda de cuál meta-heurística usar a la hora de resolver un problema concreto. Para maximizar el rendimiento del algoritmo puede ser necesaria un ajuste fino de los parámetros involucrados en el algoritmo. Por lo que es complicado comparar y evaluar a fondo las meta-heurísticas y su rendimiento.

1.1. Antecedentes y estado actual

Considerando la gran complejidad que conlleva la experimentación con meta-heurísticas han surgido herramientas con las que se puede evitar tener que implementar de cero

estos algoritmos. Estas herramientas proporcionan a los usuarios un conjunto de esqueletos o librerías para optimización de problemas en base a técnicas meta-heurísticas ya preestablecidas y extendidas en el ámbito.

Estas herramientas suponen una gran ventaja respecto a la implementación directa del algoritmo desde el principio, no sólo en términos de reutilización de código sino también en cuanto a metodología y claridad de conceptos. Sin embargo, la mayoría de estas herramientas requieren de conocimientos básicos de programación, para poder entender la estructura de clases que definen a un problema y al método de resolución en sí mismo. Por lo tanto, para utilizar estas herramientas se requiere de un conocimiento profundo sobre el problema a resolver (para poder definir correctamente la representación de las soluciones, las condiciones que hacen factibles a dichas soluciones y la función de $fitness$ que nos permitirá evaluar la calidad de las soluciones obtenidas) pero también un conocimiento específico sobre la técnica algorítmica a aplicar, su configuración y su parametrización.

Cuando experimentamos con meta-heurísticas y queremos afinar la técnica que nos ofrecerá mejores resultados tendremos que tener en cuenta algunos criterios básicos que garanticen una comparación justa:

- Probar el método con un conjunto de instancias suficientemente amplio y consolidado en la literatura.
- Sincronizar adecuadamente los parámetros del algoritmo.
- Identificar y testear el comportamiento de otros métodos para la resolución del mismo tipo de problemas/instancias, afinando los parámetros involucrados en los mismos
- Realizar un número suficiente de ejecuciones para cada uno de los métodos, problemas o instancias a evaluar.
- Evaluar la validez o la eficiencia de los distintos métodos. Para ello es necesario realizar un análisis estadístico de los datos obtenidos

Existen algunos frameworks para optimización con meta-heurísticas que ya simplifican esta experimentación. jMetal (3) es un ejemplo, proporcionando a sus usuarios un conjunto de clases que se pueden utilizar como estructura general para la resolución de problemas mediante distintas técnicas algorítmicas

1.2. Objetivos

Teniendo en cuenta los antecedentes anteriores, el objetivo del proyecto consiste en el análisis, diseño e implementación de una interfaz gráfica que permita la representación interactiva de los diferentes resultados obtenidos durante el proceso de experimentación. La interfaz debe ofrecer al usuario final un cuadro de mandos desde el cual seleccionar y configurar el tipo de análisis a realizar sobre los resultados existentes. Además, la interfaz debe ofrecer una clara separación entre la visualización del valor de las soluciones (funciones objetivos) y las soluciones en sí mismas (valores de las variables del problema).

Con el fin de diseñar e implementar esta interfaz interactiva, será necesario abordar las tareas o actividades que se relacionan a continuación:

- Actividad 1: Análisis de jMetal.
- Actividad 2: Análisis de librerías para visualizaciones web interactivas.
- Actividad 3: Diseño e implementación de una interfaz gráfica para la visualización interactiva de la calidad de las soluciones (en el espacio de las funciones objetivo).
- Actividad 4: Diseño e implementación de una herramienta para la visualización interactiva de soluciones (en el espacio de las variables de decisión).
- Actividad 5: Generación de la documentación.

Capítulo 2

Resolutores de problemas

En este capítulo se analizarán las herramientas de diseño, implementación y resolución de algoritmos jMetal y Prodef. Para conseguir un conjunto de soluciones que podremos representar mediante la interfaz desarrollada.

2.1. jMetal

2.1.1. Introducción

$jMetal$ (3) es un $framework$ para la optimización con meta-heurísticas desarrollado y mantenido por investigadores de la Universidad de Málaga que proporciona un conjunto de clases que se pueden utilizar como estructura general para la resolución de problemas mediante diferentes técnicas metaheurísticas, aprovechando la reutilización de código en las implementaciones de los operadores genéticos, las métricas o en los indicadores de calidad al compartirlos todos los algoritmos. Esto facilitaría al usuario la utilización de de las técnicas o la implementación de nuevas, así como la experimentación, la configuración y el análisis de las mismas.

2.1.2. Instalación

Para la instalación de $jMetal$ (3) primero debe instalarse las siguientes dependencias:

- Java JDK 14+
- Maven

Una vez que tenemos los requisitos mínimos podemos descargar el repositorio de su $Github$ con el siguiente comando:

```
1    $> git clone https://github.com/jMetal/jMetal.git
```

Para más información consultar la documentación (3).

2.1.3. Estructura

La estructura básica del proyecto se organiza a partir de los siguientes componentes:

Soluciones

La clase base para la configuración de soluciones en $jMetal$ es la interfaz de solución que se encuentra bajo el paquete `org.uma.jmetal.solution`. Cada solución se compone de una lista de valores objetivos, una lista de variables de decisión, un conjunto de restricciones y un mapa con los atributos. $jMetal$ también define diferentes codificaciones de las soluciones.

Entre ellas una lista de cadenas binarias, números enteros, números decimales, lista de permutaciones, lista de caracteres y una lista de soluciones compuestas en las que cada variable de solución puede tener una codificación diferente.

Algoritmos

La clase base para los algoritmos debe constar de un método `run()` para implementarla meta-heurística y retornar un resultado invocando a `getResult()`. Al ser tan básica ofrece flexibilidad a la hora de implementar meta-heurísticas, en $jMetal$ una meta-heurística es una entidad que implementa un algoritmo.

Problemas

La clase base que implementan los problemas contiene un número de variables de decisión, un número de funciones objetivo y un número de restricciones. También debe incluir un método para la evaluación de soluciones y un método de creación de soluciones.

Operadores

Un operador en $jMetal$ es una entidad que aplica un método a unos datos y devuelve un conjunto de soluciones.

Control de restricciones

$jMetal$ implementa una clase para el control de restricciones.Contiene unos métodos estáticos para la realización de este control:

- `isFeasible`: Método que nos devuelve si la solución es factible bien porque no tiene restricciones o cumple con todas las restricciones.
- `numberOfViolatedConstraints` Método que nos devuelve el número de restricciones violadas por una solución.
- `overallConstraintViolationDegree`:Método que nos devuelve el grado de violación de restricciones de una solución.
- `feasibilityRatio`:Método que nos devuelve la proporción de soluciones factibles.
- `numberOfViolatedConstraints`:Este método se utiliza para actualizar el número de restricciones violadas en una solución.
- `overallConstraintViolationDegree`:Este método se utiliza para actualizar el grado general de violación de restricciones de una solución.

2.1.4. Configuración de un estudio experimental

Para la obtención de soluciones con la herramienta $jMetal$ podemos configurar un estudio experimental.

Definición del problema

Lo primero que debemos hacer es definir el problema que queremos solucionar y implementarlo en $jMetal$. Para realizar un ejemplo usaremos uno de los problemas que ya trae $jMetal$ implementados. En concreto `One Max` donde el objetivo es maximizar el número de unos en una cadena binaria. Para eso usaremos la clase:

```
1    public class OneMax extends AbstractBinaryProblem {}
```

Configuración de los algoritmos

Una vez que hemos definido el problema que tenemos que resolver debemos configurar el estudio experimental con los algoritmos a los que queramos comparar sus soluciones y los criterios de parada a utilizar. Para ello modificaremos la clase:

```
1 public class BinaryProblemsStudy
```

- Añadimos a la lista de problemas una instancia al que nos interesa analizar.

```
1       List<ExperimentProblem<BinarySolution>> problemList = new ArrayList<>();
2       problemList.add(new ExperimentProblem<>(new OneZeroMax(512)));
3
```

- Definimos el número de ejecuciones que queremos tener. En este caso 10.

```
1       private static final int INDEPENDENT_RUNS = 10;
2
```

- Finalmente tenemos que configurar los algoritmos que queramos que resuelvan el problema. En este ejemplo usaremos el `MOCell` (algoritmo evolutivo celular multiobjetivo):

```
1       for (ExperimentProblem<BinarySolution> problem : problemList) {
2       Algorithm<List<BinarySolution>> algorithm = new MOCellBuilder<>(
3               problem.getProblem(),
4               new SinglePointCrossover(1.0),
5               new BitFlipMutation(
6                       1.0 / ((BinaryProblem) problem.getProblem()).
  bitsFromVariable(0)))
7               .setMaxEvaluations(25000)
8               .setPopulationSize(100)
9               .build();
10      algorithms.add(new ExperimentAlgorithm<>(algorithm, problem, run));
11    }
12
```

  - En esta parte podemos modificar el $crossover$, la mutación, el número de evaluaciones máximo,que será nuestro criterio de parada, y el tamaño de la población.

2.1.5. Ejecutar el experimento

Una vez que hemos configurado el experimento podemos ejecutar nuestro estudio experimental de la siguiente manera:

```
    java org.uma.jmetal.lab.experiment.studies.BinaryProblemsStudy directorio/de/salida
```

Una vez que ha concluido la ejecución del experimento se nos creará una estructura de carpetas y ficheros como la siguiente:

```
|QualityIndicatorSummary.csv
| ----data
|   ----MOCell
|       ----OneZeroMax
|               BEST_EP_FUN.csv
|               BEST_EP_VAR.csv
|               BEST_GD_FUN.csv
|               BEST_GD_VAR.csv
|               BEST_HV_FUN.csv
|               BEST_HV_VAR.csv
|               BEST_IGD+_FUN.csv
|               BEST_IGD+_VAR.csv
|               BEST_IGD_FUN.csv
|               BEST_IGD_VAR.csv
|               BEST_NHV_FUN.csv
|               BEST_NHV_VAR.csv
|               BEST_SP_FUN.csv
|               BEST_SP_VAR.csv
|               EP
|               FUN0.csv
|               FUN1.csv
|               FUN10.csv
|               FUN11.csv
|               FUN12.csv
|               FUN13.csv
|               FUN14.csv
|               FUN15.csv
|               FUN16.csv
|               FUN17.csv
|               FUN18.csv
|               FUN19.csv
|               FUN2.csv
|               FUN20.csv
|               FUN21.csv
|               FUN22.csv
|               FUN23.csv
|               FUN24.csv
|               FUN3.csv
|               FUN4.csv
|               FUN5.csv
|               FUN6.csv
|               FUN7.csv
|               FUN8.csv
|               FUN9.csv
|               GD
|               HV
|               IGD
|               IGD+
|               MEDIAN_EP_FUN.csv
```

```
|               MEDIAN_EP_VAR.csv
|               MEDIAN_GD_FUN.csv
|               MEDIAN_GD_VAR.csv
|               MEDIAN_HV_FUN.csv
|               MEDIAN_HV_VAR.csv
|               MEDIAN_IGD+_FUN.csv
|               MEDIAN_IGD+_VAR.csv
|               MEDIAN_IGD_FUN.csv
|               MEDIAN_IGD_VAR.csv
|               MEDIAN_NHV_FUN.csv
|               MEDIAN_NHV_VAR.csv
|               MEDIAN_SP_FUN.csv
|               MEDIAN_SP_VAR.csv
|               NHV
|               SP
|               VAR0.csv
|               VAR1.csv
|               VAR10.csv
|               VAR11.csv
|               VAR12.csv
|               VAR13.csv
|               VAR14.csv
|               VAR15.csv
|               VAR16.csv
|               VAR17.csv
|               VAR18.csv
|               VAR19.csv
|               VAR2.csv
|               VAR20.csv
|               VAR21.csv
|               VAR22.csv
|               VAR23.csv
|               VAR24.csv
|               VAR3.csv
|               VAR4.csv
|               VAR5.csv
|               VAR6.csv
|               VAR7.csv
|               VAR8.csv
|               VAR9.csv
|
| ----latex
|       BinaryProblemsStudy.tex
|       FriedmanTestEP.tex
|       FriedmanTestGD.tex
|       FriedmanTestHV.tex
|       FriedmanTestIGD+.tex
|       FriedmanTestIGD.tex
|       FriedmanTestNHV.tex
|       FriedmanTestSP.tex
|
| ----R
|       EP.Boxplot.R
|       EP.Wilcoxon.R
|       GD.Boxplot.R
|       GD.Wilcoxon.R
|       HV.Boxplot.R
|       HV.Wilcoxon.R
```

```
|       IGD+.Boxplot.R
|       IGD+.Wilcoxon.R
|       IGD.Boxplot.R
|       IGD.Wilcoxon.R
|       NHV.Boxplot.R
|       NHV.Wilcoxon.R
|       SP.Boxplot.R
|       SP.Wilcoxon.R
|
| ----referenceFronts
        OneZeroMax.csv
        OneZeroMax.MOCell.csv
```

Dentro de la estructura pueden encontrarse las siguientes carpetas:

- La carpeta `data` que contiene las soluciones de los diferentes algoritmos a los problemas especificados en el estudio experimental.
  - Archivos de mejores y medianas para todos los indicadores `BEST` y `MEDIAN` respectivamente.
  - Archivos con los valores de las funciones objetivo.`FUN`.
  - Archivos con los valores de las soluciones `VAR`.
- Las carpetas `latex` y `R` que contienen la exportación a LaTeX del estudio y una serie de gráficos estadísticos en R.
- Finalmente una carpeta `referenceFronts` que contiene el frente de pareto.

2.2. Prodef

2.2.1. Introducción

Prodef (1; 2) es una herramienta de resolución de meta-heurísticas que permite separar la parte lógica del diseño de un problema de su implementación. Permite a los usuarios resolver problemas de optimización complejos sin necesidad de tener conocimientos específicos de programación o algoritmia al utilizar una capa de abstracción entre el modelado del problema y la implementación de la meta heurística o algoritmo bio-inspirado a utilizar para resolverlo.

Esta herramienta fue desarrollada en un inicio por Andrés Calimero García (1) en su trabajo de final de grado. Tiene un diseño modular que le ha permitido extender sus funcionalidades en posteriores desarrollos. Sus repositorios están situados en la organización de GitHub PAL-ULL.

2.2.2. Arquitectura

Como se dijo anteriormente la herramienta está desarrollada de manera modular partiendo de un conjunto de módulos que conforman el núcleo de la aplicación, presentados en la Figura 2.1, y otra serie de módulos que expanden sus funcionalidades.

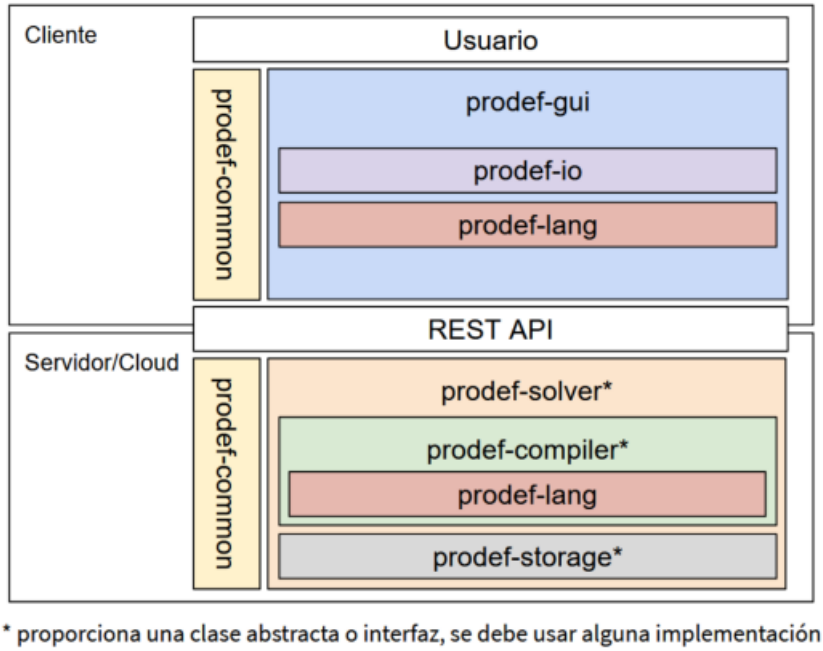


Figura 2.1: Estructura de $Prodef$ (1)

Módulos principales

- `Prodef-common`:Contiene las interfaces, tipos e implementaciones comunes para la comunicación entre el resto de módulos.
- `Prodef-solver`: Es el módulo base para los resolutores y además define la API REST que proporciona los puntos de acceso para ellos. Gestiona los ciclos de vida y llama a los resolutores cuando es necesario.
- `Prodef-compiler`: Contiene la base para todas las implementaciones de compiladores.
- `Prodef-lang`: Implementa y define el Lenguaje Específico de Dominio(DSL) que se usará para la definición de las funciones objetivo y las restricciones del problema.
- `Prodef-storage`:Define la base para todos los driver de almacenamiento no persistente.
- `Prodef-gui`:Contiene la interfaz gráfica del usuario.
- `Prodef-io`:Implementa funciones para facilitar la deserialización y validación de problemas y soluciones en formato json.

Módulos extendidos

A partir de los módulos básicos anteriormente descritos se implementan una serie de módulos que extienden de estos y expanden las funcionalidades de la herramienta y para la ejecución completa de esta se definieron una serie de módulos específicos:

- `prodef-solver-jmetal`:Módulo que define una implementación específica del resolutor de $Prodef$ para el framework $jMetal$. Da como resultado un archivo java con la definición del problema.

- `prodef-compiler-java`:Módulo que implementa un compilador para $Prodef$ basado en java.
- `prodef-storage-redis`:Implementación de un driver para bases de datos redis para el estado de las ejecuciones.
- `prodef-solver-metco`:Implementación de un resolutor de $Prodef$ para la herramienta metco.
- `prodef-compiler-cpp`: Módulo que implementa un compilador para $Prodef$ basado en c++.

2.2.3. Esquema de las soluciones

Después de una ejecución completa de la herramienta después de definir un problema y resolverlo con alguno de los resolutores podemos obtener una serie de carpetas y ficheros json que contienen las soluciones encontradas a ese problema, cuya estructura es la siguiente:

- Solver
  - Instance Type
    - Instance
      - execution1.json
      - execition2.json
    - local-config.json
- global-config.json

Los ficheros global-config.json se corresponden con la configuración global de la ejecución:

```
{
    "path":"",
    "configs":[
    {
        "algorithm": {
            "name":""
        "parameters": [],

      };
      "execution": {
        "numberOfExecutions": "",
        "printPeriod": "",
        "stopCriteria": {
          "name": "",
          "value": ""
        }
      },
      "problem": {
```

```
        "name": "",
        "objectiveName": "",
        "variant": "",
        "categories": [
          {
            "name": "",
            "numberOfInstances":""
          },
        }
    }
    ]
}
```

Contiene los siguientes campos:

- **path:** La ruta hasta la carpeta de las soluciones.
- **configs/algoritm:** Configuración del algoritmo
  - **name:** Nombre del algoritmo.
  - **parameters:** Incluye los parámetros de configuración del algoritmo
- **execution:** Incluye los parámetros de configuración de la ejecución
  - **numberOfExecutions:** Número de ejecuciones.
  - **printPeriod:** Indica cada cuanto se muestra según el criterio de parada.
  - **stopCriteria:** Criterio de parada del algoritmo. Cuando dejará de ejecutarse. Incluye el nombre del criterio de parada y su valor.
- **problem:** Definición del problema.
  - **name:** Nombre del problema.
  - **objectiveName:** Nombre del objetivo.
  - **variant:** Nombre de la variante.
  - **categories:** Categorías del problema. Incluye el nombre de la categoría y el número de instancias de esta categoría.

Los ficheros local-config.json se corresponden con la configuración de las carpetas en la que esté. Sigue la misma estructura del anterior.

Los ficheros execution.json se corresponden con los archivos que definen las soluciones:

```
{
  "executions": [
    {
      "frontSize": 1,
      "performedEvaluations": 104,
      "results": [
        {
          "constraintViolationDegree": -99344.0,
```

```
          "feasible": false,
          "objectives": [
            72150.0
          ],
          "variables": []
        }
        ]
    ,}
    ]
}
```

Tienen la siguiente estructura:

- **executions:** Configuración de las ejecuciones.
- **frontsize:** Tamaño del frente.
- **performedEvaluations** Número de evaluación.
- **results** Resultados.
  - **constraintViolationDegree:** Grado de violación de las restricciones.
  - **feasible:** Si la solución es factible o no.
  - **objectives:** Valores de las funciones objetivo.
  - **variables:** Valores de las variables

Capítulo 3

Análisis de librerías para visualización de datos

En este capítulo se identificarán y analizarán las posibles librerías a utilizar de cara al proyecto. Se han escogido para analizar tres librerías $Dash$, $Bokeh$ y $Panel$, todas ellas sirven para la visualización de datos interactivos y la implementación de cuadros de mando.

3.1. Bokeh

Bokeh es una librería de visualización de datos interactivas que permite interactuar con ellos de manera sencilla. Para su uso en aplicaciones complejas se pueden utilizar las aplicaciones de servidor de bokeh (4).

3.1.1. Instalación

Para la instalación de la herramienta introducimos en la línea de comandos lo siguiente:

```
$ > pip install bokeh
```

3.1.2. Ejemplo de uso

Una vez acabada la instalación podemos crear un archivo .py y ponernos a trabajar en él. Lo primero a realizar es importar los métodos que vamos a utilizar de la líbrería:

```
from bokeh.models import Div,ColorPicker,ColumnDataSource
from bokeh.plotting import figure, show
from bokeh.layouts import layout
```

El siguiente paso a realizar es la obtención de los datos que queramos visualizar. Para ello utilizaremos para este ejemplo el método **ColumnDataSource** de la misma librería:

```
source = ColumnDataSource(data=dict(x = [1,3,5,4,5],y=
[1,2,2,3,1]))
```

A continuación creamos el gráfico definiendo los nombres de los ejes, el tamaño del gráfico, la localización de la barra de herramientas, las herramientas que se utilizarán, un identificador para el gráfico, y un título:

```
    plot=figure( background_fill_alpha=0.2,
            plot_height=400,
            plot_width=600,
            x_axis_label='X Label',
            y_axis_label='Y Label',
            toolbar_location='right',
            tools=['save','hover','tap','pan','zoom_in','zoom_out'],
            y_range=(0,8),
            x_range=(-0.5,2),
            name="grafico1",
            title="Titulo"
            )
```

Creamos también las figuras basadas en los datos,en este caso vamos a incluir círculos por cada dato. Para ello utilizaremos el método *circle* en el que definiremos la fuente de los datos, el campo que utilizaremos para el eje x e y, una etiqueta para la leyenda, el color de relleno, la transparencia y el tamaño de las figuras.

```
    circulos = plot.circle(source=source,x='x',y='y',
    legend_label="Pruebas",fill_color="Blue",
    fill_alpha=0.5,size=60)
```

A continuación añadiremos un widget que nos permitirán cambiar el tamaño y el color de los círculos que hemos dibujado. Primero creamos un *Slider* para cambiar el tamaño:

```
    slider = Slider(start=8, end=60, value=0,step=1,
    title="Circle Size")
```

A continuación creamos un *ColorPicker* para poder elegir el color de los círculos:

```
    picker = ColorPicker(title="Circle Color")
```

Finalmente enlazamos ambos elementos con la propiedad del círculo que queremos cambiar:

```
    slider.js_link('value',circulos.glyph,'size')
    picker.js_link('color', circulos.glyph, 'fill_color')
```

También podemos decidir como se va a ver en el navegador creando una composición con *layout*:

```
    layout = layout([[div1],[picker],[plot]])
```

Finalmente podemos guardar la página web estática que nos genera automáticamente con *save* o mostrarlo en nuestro navegador directamente con *show*. Como podemos ver en la figura 3.1 y en la figura 3.2 se produce el cambio del color y tamaño que hemos elegido en los *widgets*.

```
    show(layout)
```

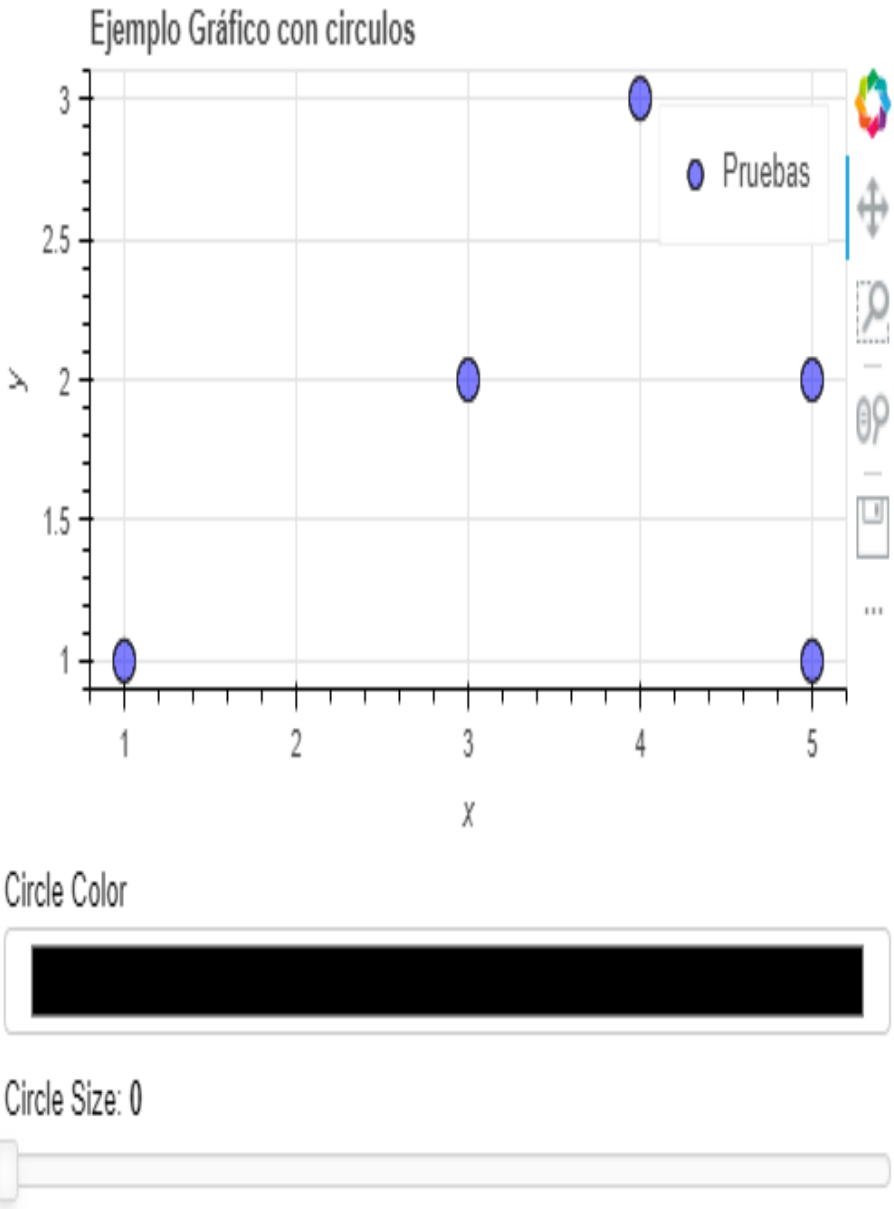


Figura 3.1: Visualización del ejemplo de Bokeh antes de los cambios.

3.1.3. Gráficos dinámicos

Para la realización de gráficos dinámicos podemos hacer uso de las aplicaciones de servidor bokeh. Para ello preparamos los datos cargando los ficheros que hemos generado con $jMetal$:

```
mediana = sorted(glob.glob(".\\data\\m-*.csv"))
mejor = sorted(glob.glob(".\\data\\b-*.csv"))
data = {
        'x_values': pd.read_csv(mediana[0],names=["x", "y"])['x'],
        'y_values': pd.read_csv(mediana[0],names=["x", "y"])['y'] }
sourceMediana = ColumnDataSource(data=data)
data = {
        'x_values': pd.read_csv(mejor[0],names=["x", "y"])['x'],
        'y_values': pd.read_csv(mejor[0],names=["x", "y"])['y'] }
sourceMejor = ColumnDataSource(data=data)
```

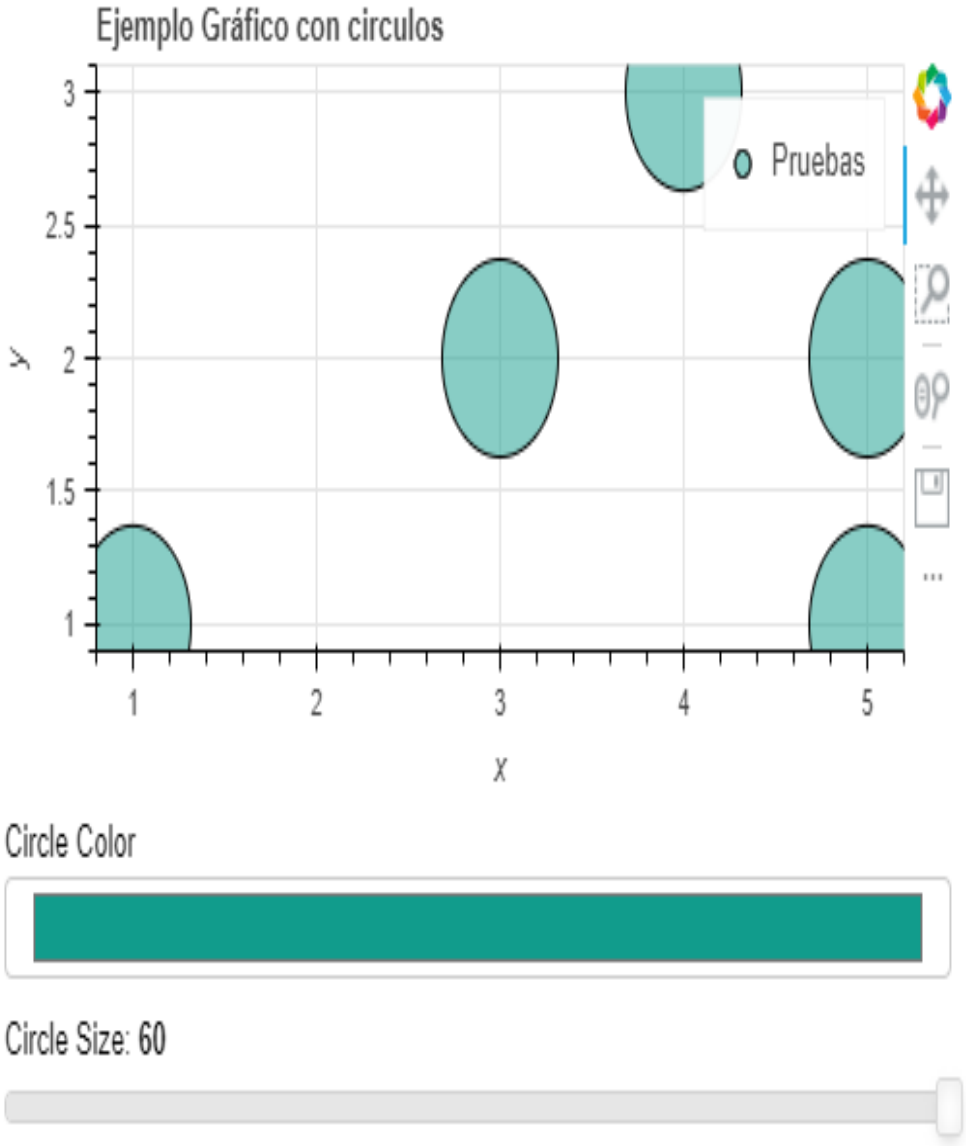


Figura 3.2: Visualización del ejemplo de Bokeh después de los cambios.

A continuación creamos una función que actualizará nuestros datos:

```
def update():
        global n_archivos,index,animation
        if (index < n_archivos-1 | index > 0):
                index = index+1
        else :
                index = 0
                if (animation!="vacio"):
                        curdoc().remove_periodic_callback(animation)
                        animation="vacio"
        sourceMediana.data = {
                'x_values': pd.read_csv(mediana[index],
                names=["x", "y"])['x'],
                'y_values': pd.read_csv(mediana[index],
                names=["x", "y"])['y']}
        sourceMejor.data = {
                'x_values': pd.read_csv(mejor[index],names=["x", "y"])['x'],
                'y_values': pd.read_csv(mejor[index],names=["x", "y"])['y']}
        plot.title.text=str(titulos[index]) + " evaluaciones"
```

Finalmente creamos la animación con $curdoc().add\_periodic\_callback()$.Y la enlazamos a un botón de inicio:

```

def start():
        global animation
        animation=curdoc().add_periodic_callback(update, refresh)
        reproducir.disabled=True
        pausa.disabled=False
        reseteo.disabled=False

reproducir = Button(label='->',width=50,aspect_ratio=1,disabled=False)
reproducir.on_click(start)
```

Una vez enlazado creamos la capa y lo introducimos en el DOM:

```
    layout = Row(plot,Column(button))
    curdoc().add_root(layout)
```

Como resultado al pulsar sobre el botón conseguiremos una animación que irá cambiando a lo largo de las evaluaciones con los datos que hemos cargado.

3.2. Dash

Dash (5) es un framework de construcción de aplicaciones de datos para python, R y Julia.

3.2.1. Instalación

Usaremos pip para instalar los paquetes necesarios:

```
    $ > pip install dash==0.21.1
    $ > pip install dash-renderer==0.13.0
    $ > pip install dash-html-components==0.11.0
    $ > pip install dash-core-components==0.23.0
    $ > pip install plotly --upgrade
```

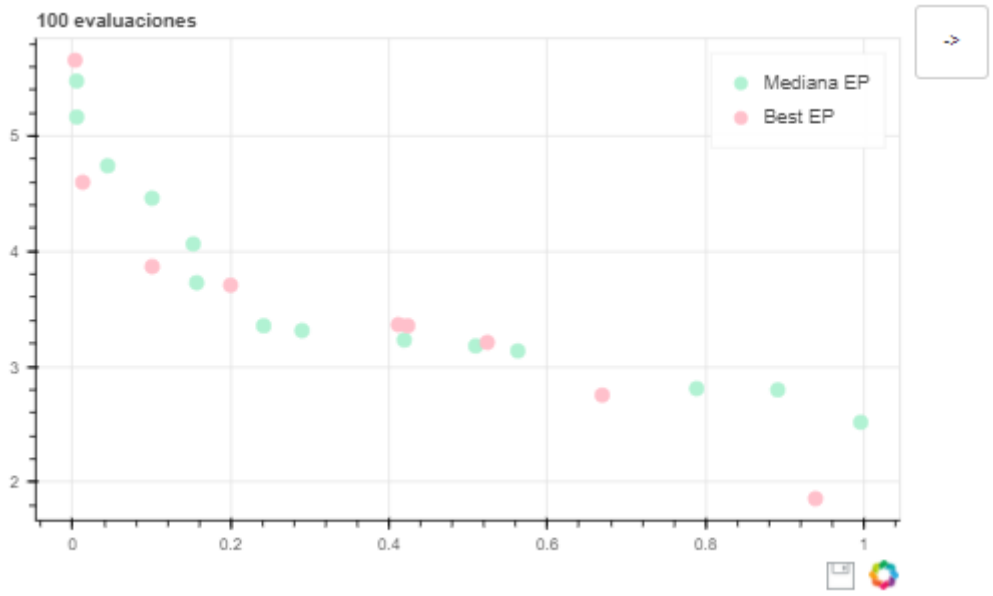


Figura 3.3: Visualización del ejemplo de Bokeh dinámico

3.2.2. Ejemplo de uso

Una vez que hemos instalado la librería podemos empezar un archivo app.py para realizar el ejemplo anterior. Comenzamos creando una instancia de la aplicación:

```
app = Dash(__name__)

```

A continuación creamos los datos que utilizaremos en este ejemplo:

```
data={'x':[1,3,5,4,5],'y':[1,2,2,3,1]}
```

Creamos también la capa que se introducirá en el DOM:

```
    app.layout = html.Div(
        children=[
            dcc.Graph(
                id="graph",
                figure={
                    "data": [
                        go.Scatter(
                            x= data["x"],
                            y= data["y"],
                            mode='markers',
                            opacity=0.8,
                            marker={
                            'size': 15,
                            'line': {'width': 0.5, 'color': 'white'}
                        }),
                    ],
                    "layout": {"title": "Ejemplo de Dash"},
                }),
            dcc.Slider(
            9,
            60,
            step=1,
            value=9,
```

```
            id='size-slider'),
            daq.ColorPicker(
            id='color-picker',
            label='Color Picker',
            value=dict(hex='#119DFF')
        ),
        ])
```

En primer lugar creamos el gráfico con el componente dcc y la función scatter de plotly. A continuación creamos un $slider$ que utilizaremos para cambiar el tamaño de los puntos y un selector de color que nos cambiará el color de los puntos.Para agregar las funcionalidades a estos elementos crearemos una callback que contendrá una salida, hacia el gráfico, y dos entradas, una para cada $widgets$:

```
@app.callback(
    Output('graph', 'figure'),
    Input('color-picker', 'value'),
    Input('size-slider','value')
)
```

Finalmente incluimos una función para la anterior callback que se encargará de actualizar el gráfico con las entradas anteriores y haremos una llamada para ejecutar el servidor:

```
    def update_figure(selected_color,selected_size):
        figure={
            "data": [
                go.Scatter(
                    x= data["x"],
                    y= data["y"],
                    mode='markers',
                    opacity=0.8,
                    marker={
                    'size': selected_size,
                    'line': {'width': 0.5, 'color': selected_color['hex']},
                    'color':selected_color['hex']
                }),
            ],
            "layout": {"title": "Ejemplo de Dash"},
        }
        return figure

    if __name__ == "__main__":
        app.run_server(debug=True)
```

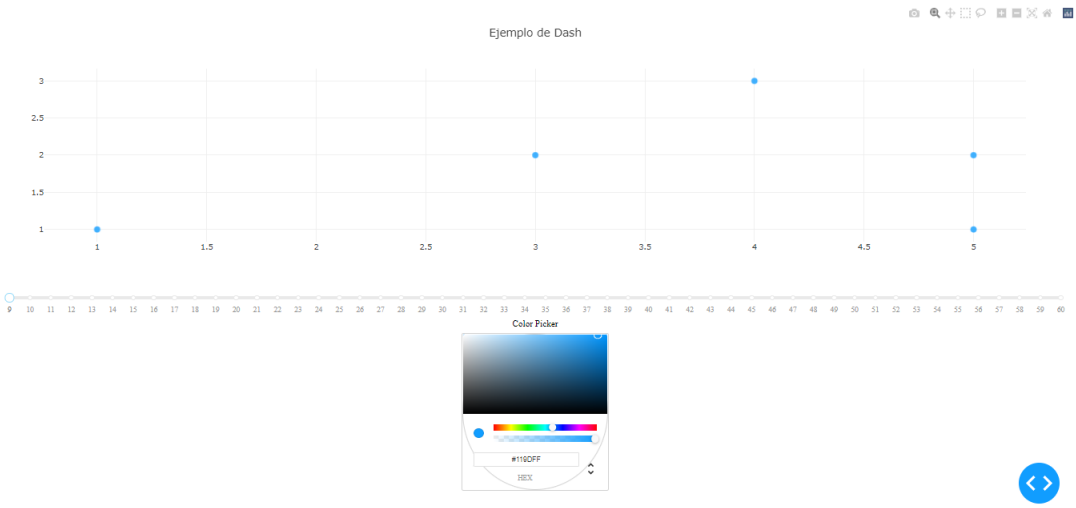


Figura 3.4: Visualización del ejemplo de Dash antes de los cambios.

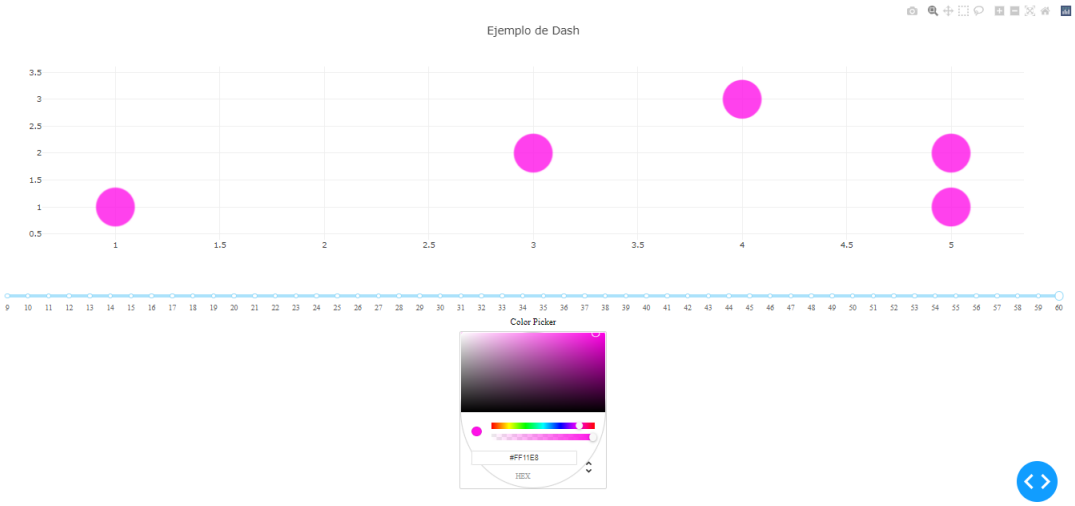


Figura 3.5: Visualización del ejemplo de Dash después de los cambios.

3.2.3. Gráficos Dinámicos

La creación de gráficos dinámicos con Dash se hace a través de las clases *Frame* y *animation* de Plotly. Primero se crean los *frames* que formarán parte de la animación:

```
for i in range(1,len(mejor)):
    frames.append(
        go.Frame(data= [
            go.Scatter(
                x= pd.read_csv(mediana[i],names=["x", "y"])['x'],
                y= pd.read_csv(mediana[i],names=["x", "y"])['y'],
                mode='markers',
                opacity=0.8,
                marker={
                'size': 15,
                'line': {'width': 0.5, 'color': 'pink'},
                'color':'pink'
            }),
            go.Scatter(
                x= pd.read_csv(mejor[i],names=["x", "y"])['x'],
```

```
                y= pd.read_csv(mejor[i],names=["x", "y"])['y'],
                mode='markers',
                opacity=0.8,
                marker={
                'size': 15,
                'line': {'width': 0.5, 'color': 'green'},
                'color':'green'
            })
        ]))
```

Creamos también la figura inicial que contendrá los datos iniciales, un botón que iniciará la reproducción de la animación y el objeto $frames$:

```
fig = go.Figure(
    data= [
            go.Scatter(
                x= pd.read_csv(mediana[0],names=["x", "y"])['x'],
                y= pd.read_csv(mediana[0],names=["x", "y"])['y'],
                mode='markers',
                opacity=0.8,
                marker={
                'size': 15,
                'line': {'width': 0.5, 'color': 'pink'},
                'color':'pink'
            }),
            go.Scatter(
                x= pd.read_csv(mejor[0],names=["x", "y"])['x'],
                y= pd.read_csv(mejor[0],names=["x", "y"])['y'],
                mode='markers',
                opacity=0.8,
                marker={
                'size': 15,
                'line': {'width': 0.5, 'color': 'green'},
                'color':'green'
            })
    ],
    layout=go.Layout(
        title="Start Title",
        updatemenus=[dict(
            type="buttons",
            buttons=[dict(label="Play",
                          method="animate",
                          args=[None])])]
    ),
    frames=frames
)
```

Finalmente añadimos el objeto a una capa y agregamos la ejecución del servidor:

```
app.layout = html.Div(
    children=[
        dcc.Graph(id="Graph",figure=fig)
    ])

if __name__ == "__main__":
    app.run_server(debug=True)
```

3.3. Panel

Panel (6) es una librería de python que utiliza y simplifica el uso de los Bokeh servers, haciendo de esta una herramienta potente que facilita la creación de widgets y la personalización de herramientas para nuestros cuadros de mando.

3.3.1. Instalación de la librería

Podemos instalar la librería usando pip:

```
    $ > pip install panel
```

3.3.2. Ejemplo de uso

Primero importamos las librerías de panel y bokeh:

```
import panel as pn
from bokeh.plotting import figure, show, save
from bokeh.layouts import gridplot
from bokeh.models import ColumnDataSource
from panel.interact import interact
pn.extension()
```

Luego preparamos los datos y el gráfico con *bokeh*:

```
    source = ColumnDataSource(data=dict(x = [1,3,5,4,5],y=
    [1,2,2,3,1]))
    plot = figure(title="Example", x_axis_label='x',
    y_axis_label='y')
    circulo= plot.circle(x="x", y="y", size=10, source=source)

```

Panel trae algunos cambios en su uso respecto a *bokeh*, como el uso de un decorador(@interact) para crear los *widgets* de manera sencilla.

De forma automática creamos un *slider* mediante este decorador:

```
    @interact(x=(10,80))
    def sizes(x):
        circulo.glyph.size=x
```

Creamos también una lista para la elección de color de los puntos:

```
    @interact(x={"Green","Blue","Pink","Red"})
    def colores(x):
        circulo.glyph.fill_color=x

```

Finalmente lanzamos la aplicación con:

```
     pn.serve(app)

```

Como podemos ver a continuación en la figura 3.6

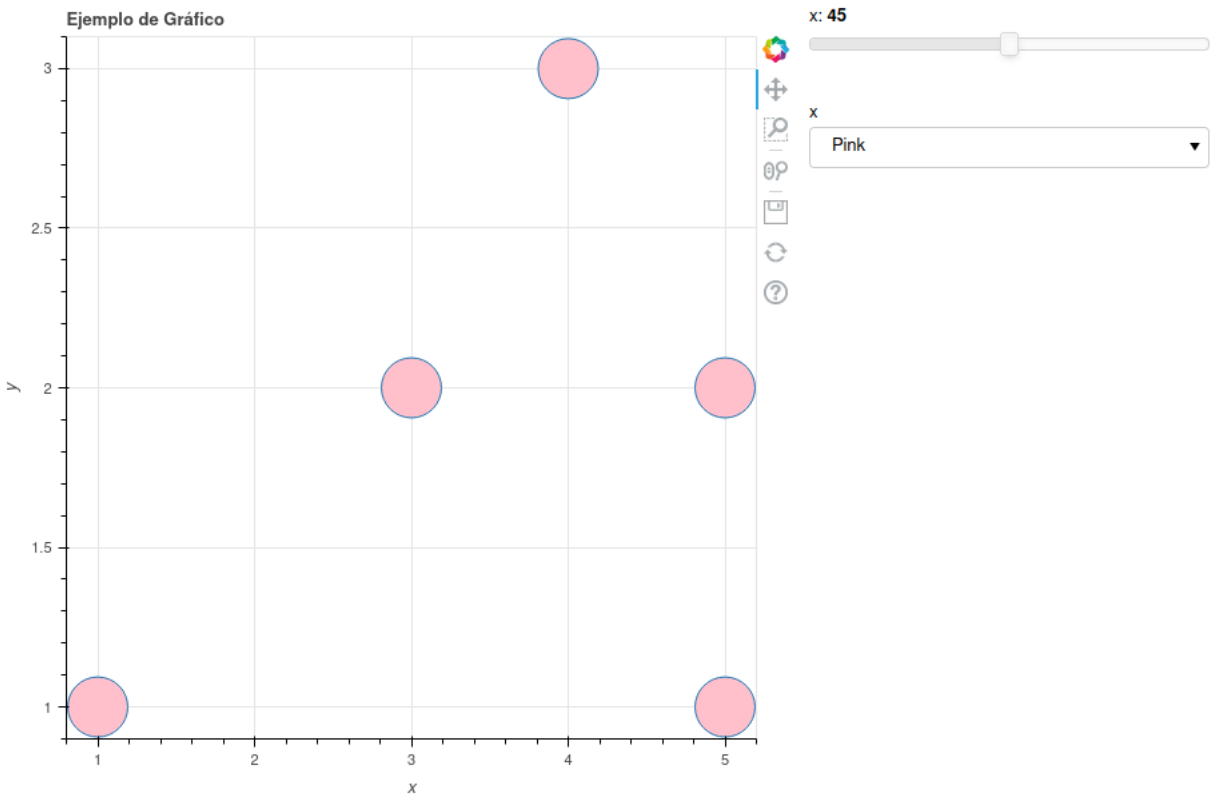


Figura 3.6: Visualización del ejemplo de Panel

3.4. Conclusiones

Para este proyecto la librería elegida ha sido *Bokeh* debido a su facilidad de implementación y uso. Las ventajas de esta librería frente a *Panel* y *Dash* son las siguientes:

- Mayor flexibilidad a la hora de crear gráficos interactivos para visualizaciones de datos personalizadas al basarse en primitivas de bajo nivel. Mientras que *Dash* se enfoca en crear paneles de control y Panel para visualizaciones de datos menos complejas.
- Bokeh y Panel son más fáciles de utilizar al proporcionar herramientas de alto nivel para creación visualizaciones interactivas que Dash
- Bokeh tiene integración sencilla con otra librerías de python tales como Pandas y Numpy.
- Bokeh posee una documentación más completa que *Dash* o *Panel* a la hora de aprender a usarlo.

Capítulo 4

Interfaz Gráfica para la visualización interactiva de la calidad de las soluciones

4.1. Tecnologías Utilizadas

El lenguaje principal utilizado para la realización de este proyecto es $Python$ junto a una de las librerías de visualización de datos analizadas en el Capítulo 3 $Bokeh$. También se hace uso del lenguaje $Javascript$ para el uso de los botones de control /reflib:.

Se utiliza pip para el manejo e instalación de paquetes para $python$ y $pandas$ para la creación de las estructuras de datos.A continuación se muestran las versiones de los principales paquetes instalados:

Tecnología	Versión
Python	V 3.11.3
bokeh	V 2.4.3
imageio	V 2.28.1
numpy	V 1.24.3
pandas	V 2.0.1
pip	V 22.3.1
selenium	V 4.9.1

Tabla 4.1: Versiones de las dependencias

- $Bokeh$: es una librería de creación de gráficos interactivos y dinámicos.
- $json$: es utilizada para la lectura de los archivos json que contienen los datos a visualizar.
- $base64$: es utilizada para decodificar el fichero que sube el usuario a la aplicación.
- $glob$: carga los ficheros con imágenes para la creación del gif.
- imageio: es una librería de manipulación de imágenes. Es utilizada para la creación de un gif con la evolución de los algoritmos.
- $math$: utilizada para obtener el número pi.

- *random*: es utilizada para elegir al azar la forma y el color de las figuras que se crearan en el gráfico.
- *selenium*: utilizado para la interacción con el navegador web.

4.2. Instalación y ejecución

El código fuente puede obtenerse en el siguiente repositorio https://github.com/carolalvarezm/TFG Para clonarlo utilizamos:

```
$ git clone https://github.com/carolalvarezm/TFG.git
```

Una vez clonado debemos instalar las dependencias de la siguiente manera:

```
$ pip install bokeh==2.4.3
$ pip install imageio==2.28.1
$ pip install numpy==1.24.3
$ pip install pandas==2.0.1
$ pip install panel==0.14.4
$ pip install selenium==4.9.1
```

También podemos hacer uso del makefile proporcionado:

```
install:
    pip -r install paquetes.txt
run:
bokeh serve .
```

```
$ make install
```

Y del archivo .bat(install.bat) proporcionado para los entornos windows, haciendo doble click sobre este. Cuando ya tengamos instaladas las dependencias podemos hacer uso de un servidor bokeh para desplegarlo. Ubicándonos en la misma carpeta del código fuente:

```
$ bokeh serve .
```

O haciendo uso, en un entorno linux, del makefile proporcionado:

```
$ make run
```

4.3. Arquitectura

La estructura del proyecto está formada por una serie de clases y un archivo principal main.py que contiene el programa principal.

4.3.1. Clase Figura

Esta clase contiene y inicializa los datos necesarios para dibujar una única ejecución de una meta-heurística sobre el gráfico. Para instanciarla le debemos pasar los siguientes datos:

- dataPath: la ruta hasta el fichero de configuración de esta instancia.
- printPeriod: la frecuencia de impresión o muestra de las soluciones.

- stopCriteria: Criterio de parada usado para terminar la ejecución del algoritmo de optimización.
- stopCriteriaValue: Valor del criterio de parada.
- executionsNumber ; Número de ejecuciones.
- instance: Instancia.
- problemName: Nombre del problema.
- problemSize; Tamaño del problema.
- color: El color que tendrá al dibujarse sobre el gráfico.
- shape: La forma que tendrá al dibujarse sobre el gráfico.

Esta clase se encarga también de traducir a un objeto python los ficheros de .json de solución de esa instancia del problema.

```
    from bokeh.models import ColumnDataSource
    import jso
    class Figura():
        def __init__(self,dataPath,printPeriod,stopCriteria,stopCriteriaValue,
    executionsNumber,instance,problemName,problemSize,color,shape):
            ...
        def loadData(self,datapath):
            for i in datapath:
                file=open(i)
                dataJson=json.loads(file.read())
                self.data.append(dataJson)
```

4.3.2. Clase Plot

En esta clase definimos la base del gráfico, asignando, el título, los ejes, el tamaño y las herramientas que se tendrán disponibles:

```
def __init__(self,x_axis_label):
        self.plot=figure(title="title", x_axis_label=x_axis_label, y_axis_label="
    y_axis_label",tools=['save','hover','tap','pan'],toolbar_location="below",plot_width
    =1000, plot_height=800)
        self.legendInstances=""
```

También se añade la información para los tooltips del hover, de manera que al pasar por encima de algún punto del gráfico salga la información que queremos:

```
        self.hover = self.plot.select(dict(type=HoverTool))
        self.hover.tooltips = [("Execution", "@Execution"), ("Objective Value", "
    @ObjectiveValue"),("Feasibility","@feasibility"),("Solver","@Solver"),("Instance Type
    :","@InstanceType"),("Instance","@Instance")]
```

Se definen además las funciones para crear la leyenda, ocultarla y mostrarla y modificar el título que se ejecutarán cuando interactuemos con los controles de la página.

```
def updateTitle(self,title)
...
def createLegends (self,nombresInstancias,nombresCategorias,nombresProblemas,problems,
    figuras):
...
def ocultarLeyenda(self):
...
def mostrarLeyenda(self):
...
def ocultarItemLeyenda(self,attr):
```

4.3.3. Clase Mono-objetivo

En esta clase se ha implementado todo los métodos necesarios para mostrar y controlar las visualizaciones de las soluciones de los problemas con una única función objetivo.Cada instancia de esta clase contendrá todas las soluciones de las ejecuciones de cada instancia de un problema. Extiende de la clase figura comentada anteriormente y define los siguientes métodos:

- createData():Crea los datos que se han cargado de los ficheros de solución .json.Inicializa los $dataframe$ que se usarán a lo largo de la aplicación. Carga las soluciones, se seleccionan y guardan por separado las soluciones factibles, se calculan los máximos, mínimos, cuantiles, y desviación estándar para ese conjunto de soluciones al problema.
- updatefeasibility(): Al llamarlo modifica si se muestran sólo las soluciones factibles o todas.
- removeFigure(): modifica la visibilidad de las figura para este problema a no visible.
- addFigure(): modifica la visibilidad de las figura para este problema a visible.
- updateMarker():Modifica la forma de mostrarse sobre el gráfico.
- updateColor():Modifica el color a mostrar sobre el gráfico.
- updatePlot():Cambia los datos del gráfico por ejecuciones a aquellos de la evaluación que le pasamos en el index.
- updatemaxplot():Cambia los datos del gráfico de estadísticas a aquellos de la evaluación que le pasamos en el index.
- updateboxplot():Cambia los datos del gráfico de boxplot a aquellos de la evaluación que le pasamos en el index.
- createBoxplot():Crea el diagrama de cajas y bigotes.

4.3.4. Clase Multi-objetivo

En esta clase se ha implementado todo los métodos necesarios para mostrar y controlar las visualizaciones de las soluciones de los problemas con múltiples funciones objetivo.Cada instancia de esta clase contendrá todas las soluciones de las ejecuciones de cada

instancia de un problema. Extiende de la clase figura comentada anteriormente y define los métodos anteriormente nombrados de la clase monoobjetivo además de los siguientes:

- puntosProximos(old,new):Calcula el punto más proximo para todas las soluciones de una evaluación anterior en todas las de la nueva evaluación para la animación del algoritmo.
- puntosMedios(new, old, cercanos):Calcula el punto medio para todas las soluciones de una evaluación anterior en todas las de la nueva evaluación, para suavizar la animación.
- calculoDistancia(puntoA,puntoB):Calcula la distancias entre dos soluciones dadas.

4.3.5. Clase Config

Contiene los controles para el funcionamiento de los botones de la interfaz. Utiliza $Bokehjs$ para definir el comportamiento de estos. Contiene los botones de problemas, variantes y categorías del experimento, máximo, mínimo y media, mostrar sólo soluciones factibles y el del cambio de color.Ver figura 4.1 Todos ellos ,excepto el de cambio de color, funcionan como un interruptor si apretamos una vez se ocultarán los elementos a los que hacen referencia y si volvemos a apretarlo se mostraran de nuevo.

4.3.6. Estructura de soluciones

Para la introducción de datos de las soluciones se utilizan una serie de ficheros .json que definen dentro de la estructura de carpetas de la solución la configuración de estas. La estructura que debe seguir el fichero de configuración y los ficheros de las ejecuciones es el mismo que se definió para $Prodef$. Como se comentaba en el capítulo 2 $Prodef$ define una estructura[(2)] de carpetas y ficheros .json. El fichero global-config.json contiene la configuración global del experimento y su definición es el punto de entrada de la interfaz gráfica. Los ficheros execution.json incluyen las soluciones de una ejecución del problema, separadas entre sí por el criterio de parada estos contienen los datos de la solución a representar en las gráficas a través de las animaciones en las gráficas.

4.3.7. Main.py

Este fichero contiene la lógica principal de la aplicación.El ciclo de ejecución es el siguiente:

Se añade al DOM las pestañas vacías

Para añadir las pestañas al DOM se deben definir los $layout$ y luego los paneles necesarios. En este primer momento se mantienen vacíos excepto el de $fileinput$ que contendrá un campo de selección de fichero y una ayuda para el usuario. Finalmente hacer uso de $curdoc.add_root()$ para añadirlos a la raíz:

```
divh1=Div(text="""<h1>Upload a Configuration File</h1>""")
h2fileinput=Div(text="""<h2>Example of a configuration file:</h2>""")
h2fileinput2=Div(text="""<h2>Usage of a configuration file:</h2>""")
divfileinput=Div(text="""
<pre>{
```

Figura 4.1: Vista de los botones de configuración

```
"path":"",
"configs":[
{
    "algorithm": {
        "name":""
    "parameters": [],

  };
  "execution": {
    "numberOfExecutions": "",
    "printPeriod": "",
    "stopCriteria": {
      "name": "",
      "value": ""
    }
  },
  "problem": {
    "name": "",
    "objectiveName": "",
    "variant": "",
    "categories": [
```

```
          {
            "name": "",
            "numberOfInstances":
          },
        }
    }
    ]
}</pre>
""")
divfileinput2=Div(text="""
The global-config.json files correspond to the global configuration of the execution. It contains the following fields:
<ul>
    <li><strong>path:</strong> The path to the solutions folder.</li>
    <li><strong>configs/algorithm:</strong> Algorithm configuration
        <ul>
            <li><strong>name:</strong> Algorithm name.</li>
            <li><strong>parameters:</strong> Includes the algorithm configuration parameters.</li>
        </ul>
    </li>
    <li><strong>execution:</strong> Execution configuration parameters
        <ul>
            <li><strong>numberOfExecutions:</strong> Number of executions.</li>
            <li><strong>printPeriod:</strong> Indicates how often it is displayed according to the stop criteria.</li>
            <li><strong>stopCriteria:</strong> Algorithm’s stopping criteria. When it will stop running. Includes the name of the stop criteria and its value.</li>
        </ul>
    </li>
    <li><strong>problem:</strong> Problem definition.
        <ul>
            <li><strong>name:</strong> Problem name.</li>
            <li><strong>objectiveName:</strong> Objective name.</li>
            <li><strong>variant:</strong> Variant name.</li>
            <li><strong>categories:</strong> Problem categories. Includes the category name and the number of instances in that category.</li>
        </ul>
    </li>
</ul>""")

controllayout=layout(children=[],sizing_mode="stretch_width", width=800)
controllayout2=layout(children=[])
controllayout3=layout(children=[])
controllayout4=layout(children=[])

tab1 = Panel(child=controllayout, title="By Executions",disabled=True)
tab2 = Panel(child=controllayout, title="Evolution",disabled=True)
tab3 = Panel(child=controllayout, title="Boxplot",disabled=True)
tab4 = Panel(child=column(divh1,fileInput,row(column(h2fileinput,divfileinput),column(h2fileinput2,divfileinput2))),title="File Input")
tab5= Panel(child=controllayout, title="Data",disabled=True)

layout=Tabs(tabs=[tab4,tab1,tab2,tab3,tab5],width=800)

curdoc().add_root(layout)
```

Se cargan los datos del fichero de configuración

Cuando el campo de selección de fichero de la primera pestaña es modificado se llama a la función $fileUpdate$.Esta función se encarga de cargar los datos desde el fichero de configuración a los gráficos:

```
            fileInput=FileInput(accept=".json")
            fileInput.on_change('value',fileUpdate)

```

- Dentro de esta función se decodifica el fichero de entrada y se convierte el json entrante en un objeto:

```
            GlobalConfig = base64.b64decode(fileInput.value).decode('UTF-8')
            config=json.loads(GlobalConfig)

```

- A continuación, se instancian los diferentes tipos de gráficos(Ejecuciones, Evolución y diagrama de cajas):

```
            plot=Plot("Execution Number")
            maxplot=Plot("Evaluations")
            boxplot=figure(x_range=[''],tools=['save','hover','tap','pan','
    ywheel_zoom'],width=800)

```

- Agregamos el texto que se puede ver al final de la figura 4.4 como capa sobre los gráficos de ejecuciones y el diagrama de cajas:

```
                text_source = ColumnDataSource({'evaluaciones': ['0']})
                text        = Text(
                  x=2,y=32,
                  text='evaluaciones',
                  text_font_size='150pt',
                  text_color='#EEEEEE',
                  name="evolutiontext"
                  )
    plot.plot.add_glyph(text_source,text)
    boxplot.add_glyph(text_source,text)

```

- Se recorre la estructura del json y se instancia la clase MonoObjetivo o MultiObjetivo según sea el caso con los datos de la configuración.

4.4. Vistas de la interfaz gráfica

La interfaz gráfica una vez la hemos desplegado se divide en una serie de pestañas como podemos ver en la figura 4.2. En un inicio sólo se se encuentra desbloqueada la primera, que es en la que cargamos el fichero de configuración del directorio de datos. Una vez que lo hemos cargado podremos desplazarnos por el resto de las pestañas. La estructura de las pestañas es la siguiente:

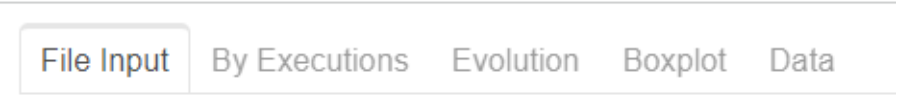


Figura 4.2: Vista de las pestañas o paneles de la interfaz

4.4.1. File Input

En esta vista se carga el fichero de configuración que contiene todas las especificaciones para acceder a las carpetas: Para cargarlo hacemos click sobre el botón de seleccionar

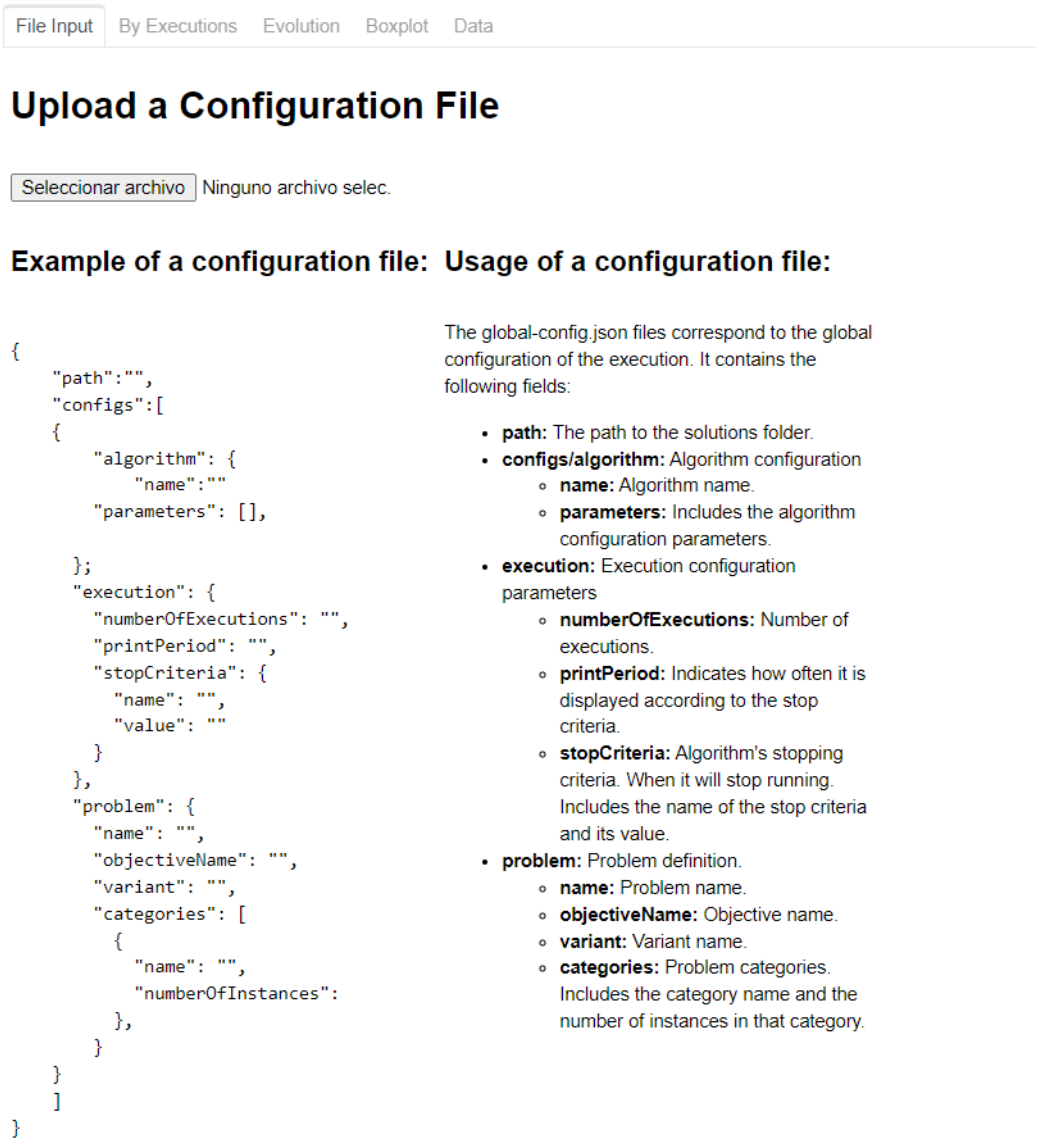


Figura 4.3: Vista de la pestaña file input

archivo, y cuando se nos abra el cuadro de diálogo elegimos el fichero de configuración de nuestro experimento. Debe tener la estructura siguiente:

```
{
    "path":"",
    "configs":[
    {
```

```
        "algorithm": {
            "name":""
        "parameters": [],

      };
      "execution": {
        "numberOfExecutions": "",
        "printPeriod": "",
        "stopCriteria": {
          "name": "",
          "value": ""
        }
      },
      "problem": {
        "name": "",
        "objectiveName": "",
        "variant": "",
        "categories": [
          {
            "name": "",
            "numberOfInstances":""
          },
        }
    }
    ]
}
```

4.4.2. Executions Plot

En esta pestaña está el gráfico de ejecuciones, en él se muestran todas las soluciones en función del número de la ejecución. Se cargan sólo los datos correspondientes a las primeras soluciones.

Reproductor

Para ver la evolución del algoritmo en función de la visualización podemos utilizar el reproductor 4.5.Este reproductor nos sirve para manejar los datos que se mostraran sobre el gráfico permitiéndonos ver una animación de las soluciones a lo largo de las sucesivas evaluaciones haciendo click en el botón de reproducir 4.5. También contiene un botón de pausa y uno de stop para pausar o terminar la animación antes de que llegue al final. Se incluyen también unos botones de hacia delante y hacia atrás 4.6 par actualizan los datos moviéndose hacia los datos de la anterior o siguiente evaluación. Finalmente tenemos dos slider uno que se mueve a lo largo del ciclo de la animación según pasan las evaluaciones 4.7 y que podemos utilizar para ir hasta una evaluación concreta y el slider de velocidad que podemos ver en la figura 4.8 que sirve como su nombre indica para cambiar la velocidad de la animación. Para montar el reproductor se ha hecho de la siguiente manera:

Figura 4.4: Vista del gráfico de ejecuciones

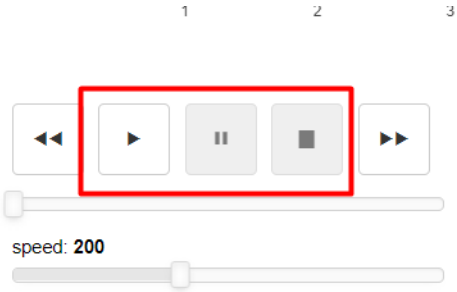


Figura 4.5: Vista del reproductor. Play, Stop y Pause

- Primero definimos todos los botones:

```
        reproducir = Button(label='',width=50,aspect_ratio=1,disabled=False)
        reproducir.on_click(start)
        pausa = Button(label='',width=50,aspect_ratio=1,disabled=True)
        pausa.on_click(stop)
        reseteo = Button(label='',width=50,aspect_ratio=1,disabled=True)
        reseteo.on_click(reset)
        siguiente = Button(label='',width=50,aspect_ratio=1)
        siguiente.on_click(next)
        previo = Button(label='',width=50,aspect_ratio=1)
        previo.on_click(previous)
        velocidad= Slider(start=10, end=500, value=200,step=1,title="speed")
        velocidad.on_change('value', slider_velocidad)
        reproductor=[previo,reproducir,pausa,reseteo,siguiente]

```

- Definimos también las funciones de cada uno de los controles.

```
        def update():
        global numberofplots,index,animation,text_source,slider,evaluaciones
        if (index < numberofplots-1 and index >= 0):
                index = index+1
                slider.value=evaluaciones[index]
        else :
            index = 0
```

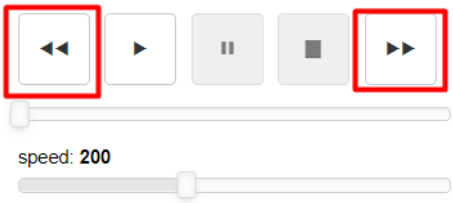


Figura 4.6: Vista del reproductor. Atrás y adelante

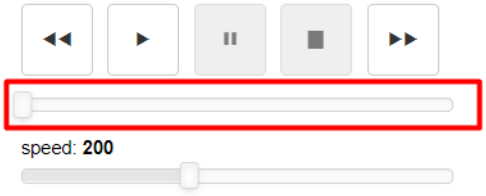


Figura 4.7: Vista del slider de evaluaciones

```
            pausa.disabled=True
            reproducir.disabled=False
            reseteo.disabled=True
            if (animation!=None):
                    curdoc().remove_periodic_callback(animation)
                    animation=None
            slider.value=evaluaciones[index]

        def slider_update(attrname, old, new):
                global index
                index=evaluaciones.index(slider.value)
                for i in problems:
                    for j in i:
                        for k in j:
                            k.updatePlot(index)
                            k.updateboxplot(index)
                text_source.data={'evaluaciones':[str(slider.value)]}
        def start():
                global animation
                animation=curdoc().add_periodic_callback(update, refresh)
                reproducir.disabled=True
                pausa.disabled=False
                reseteo.disabled=False

        def stop():
                global animation
                if (animation!=None):
                        curdoc().remove_periodic_callback(animation)
                        animation=None
                reproducir.disabled=False
                pausa.disabled=True
        def next():
                update()
        def previous():
                global index
                index=index-2
```

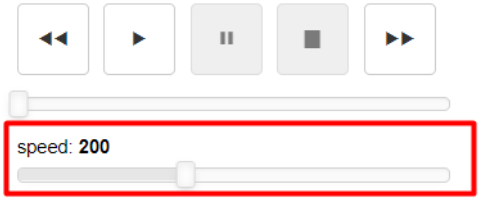


Figura 4.8: Vista del slider de velocidad

```
                update()
        def reset():
                global index
                index=-1
                update()
                pausa.disabled=True
                reproducir.disabled=False
                reseteo.disabled=True
        def slider_velocidad(attrname, old, new):
            global refresh
            refresh=velocidad.value

```

- Definimos los valores iniciales de las variables index y refresh en un inicio a 0. Se utilizarán para el índice de la evaluación por la que va la animación y la velocidad de la animación:

```
        index=0
        refresh = 200

```

Setting

A través del panel de control en la pestaña "Setting"podemos modificar lo que visualizamos en el gráfico. Este panel tiene una estructura como la que podemos observar en la figura 4.1. Primero contiene botones para cada uno de los problemas, categorías y variantes, estos botones son interruptores de visualización de las figuras en el gráfico si por ejemplo dejamos marcados sólo los botones del primer problema y una de las categorías como en la figura 4.14 podemos observar se han ocultado todas las demás figuras del gráfico. En el gráfico se distinguen además las soluciones factibles, las que no violan las restricciones del problema, de las no factibles poniendo las primeras como rellenas y las segundas como sólo con la línea. Podemos filtrar por estas con el botón de Only feasible solutions4.9 También hay un botón para mostrar o ocultar la leyenda , otro para exportar la animación del gráfico a gif 4.10 y finalmente un botón para ocultar el texto del gráfico que contiene el número de la evaluación actual 4.11.

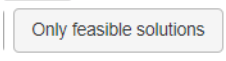


Figura 4.9: Botón de Sólo soluciones factibles

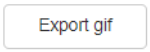


Figura 4.10: Botón de exportar a gif

Figura 4.11: Botón "Hide text"

Customize

También se ha incluido una sección para personalizar el gráfico, podemos verla en la figura 4.12. Mediante esta podemos cambiarle la figura a uno de los problemas, cambiar el color a una de las variantes, cambiar el título del gráfico y el de los ejes .

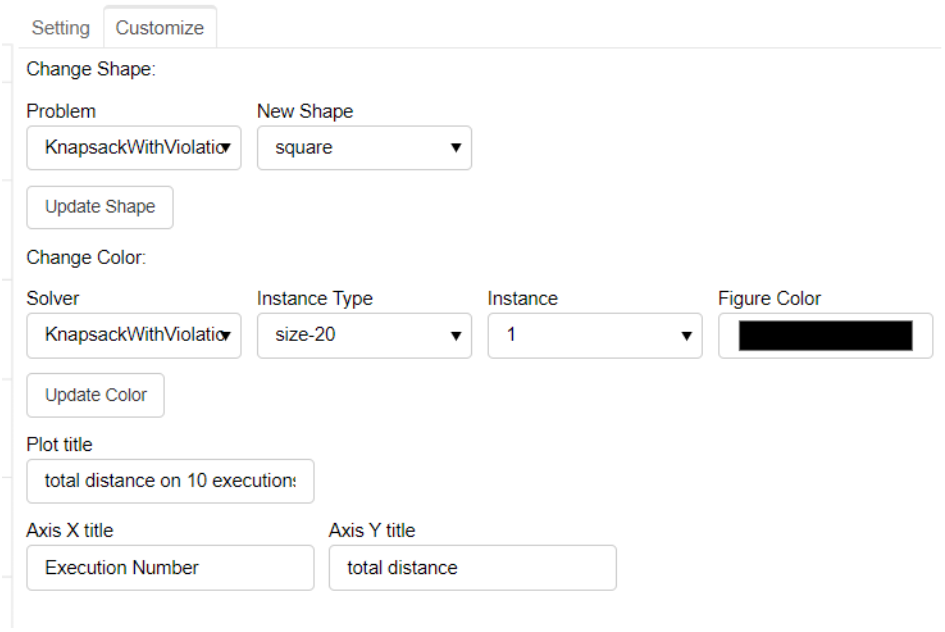


Figura 4.12: Panel con las personalizaciones del gráfico

4.4.3. Evolution Plot

El gráfico de evolución muestra los máximos, mínimos y medias de cada una de las soluciones a lo largo de las evaluaciones. Como podemos ver en la figura 4.13 hay algunos de los marcadores que están rellenos y otros vacíos, tal y como pasa con los marcadores del gráfico de ejecuciones representan las funciones factibles y no factibles respectivamente.

Setting

Como en el gráfico anterior podemos modificar lo que visualizamos a través de este panel 4.14. La parte que se añade con respecto al anterior son unos botones para visualizar o ocultar por separado estos y cambiar el número de indicadores(ticker).

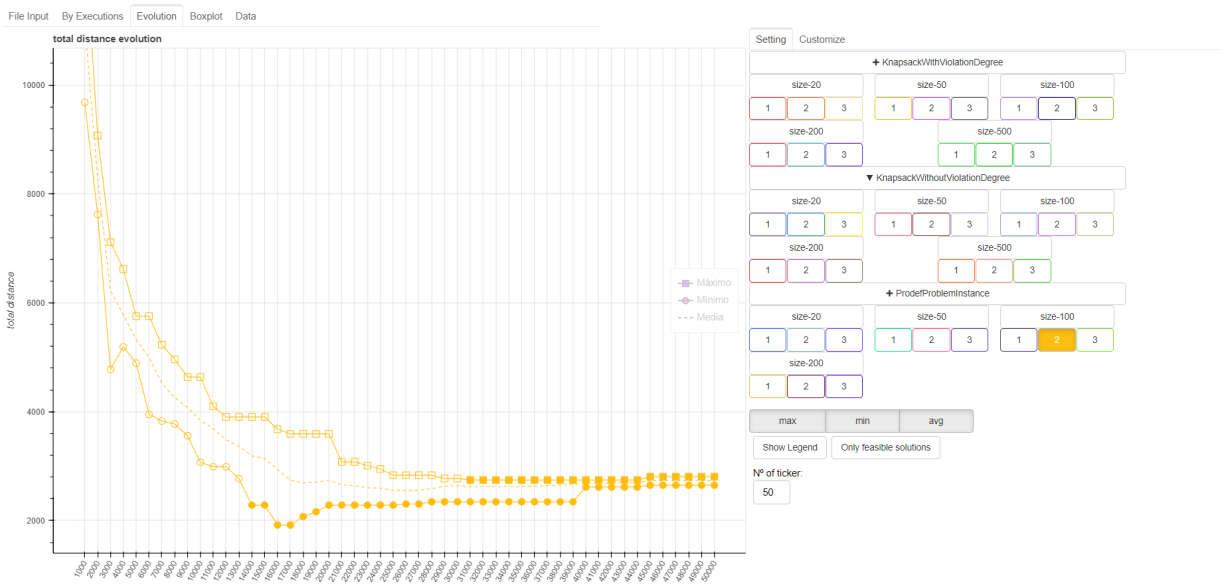


Figura 4.13: Vista del gráfico de evolución

Customize

Dentro de este apartado podemos cambiar lo mismo que en el gráfico anterior excepto por el cambio de la forma.

4.4.4. Box plot

El último gráfico es el del diagrama de cajas y bigotes que nos proporciona información sobre la distribución de los datos y la identificación de valores atípicos. Este gráfico lo construimos trazando una caja que abarca desde el primer al tercer cuartil, dibujando en medio de estos una línea donde se encuentre la mediana. Finalmente debemos extender los bigotes hasta los valores máximos y mínimos. Como podemos ver en la figura 4.15 se construyen con los datos de las soluciones.

Setting

Si queremos visualizar sólo una parte de los diagramas en primer lugar debemos seleccionar los que queremos ver.En segundo lugar, debemos hacer click en üpdate boxplot"para centrar la vista en los seleccionados 4.16. Además en este gráfico podemos ver también una animación haciendo uso del reproductor con los diferentes datos de las evaluaciones. También se incluye una opción para sólo mostrar las soluciones factibles, en la que se recalculan los diagramas sólo con soluciones factibles. Y finalmente una opción para exportar la animación a gif.

4.4.5. Data table

Finalmente se implementa la vista *Datatable* 4.17 en la que se muestra una tabla con los resultados cargados. Esta tabla podemos filtrarla mediante el widget *Multichoice* que implementa *Bokeh* estos se han adaptado para filtrar por problema, categoría, variante o evaluaciones. Mediante este filtro pueden elegirse múltiples opciones de cada uno de los seleccionables como podemos ver en la figura 4.18 .

Figura 4.14: Vista de la configuración del gráfico de evolución

Mediante la función $Dataframe.to_latex$ de $Pandas$ pasamos el dataframe una vez filtrado para que nos lo convierta en una tabla de LaTeX. Haciendo click en el botón "Export to LaTeX"se llamará a esta función y se nos descargará un fichero con la tabla en LaTeX.

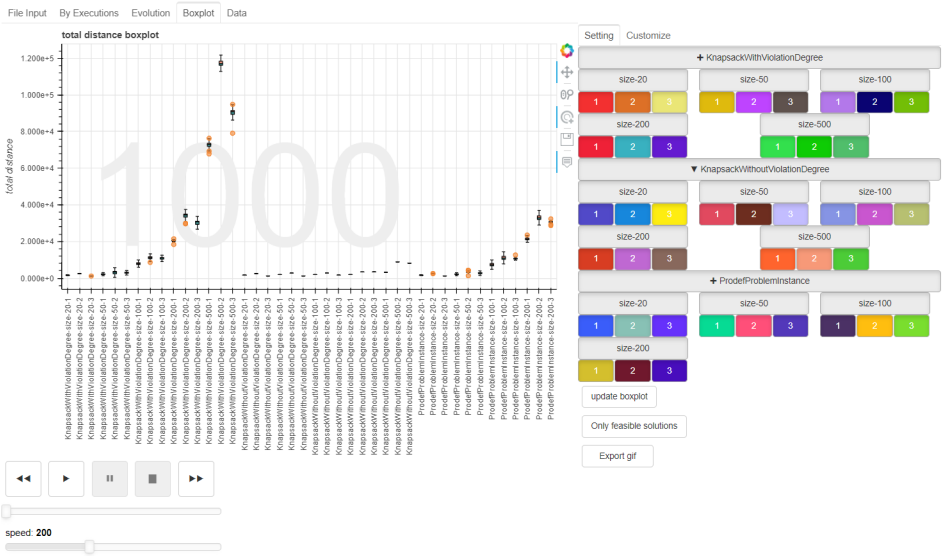


Figura 4.15: Vista del diagrama de cajas y bigotes

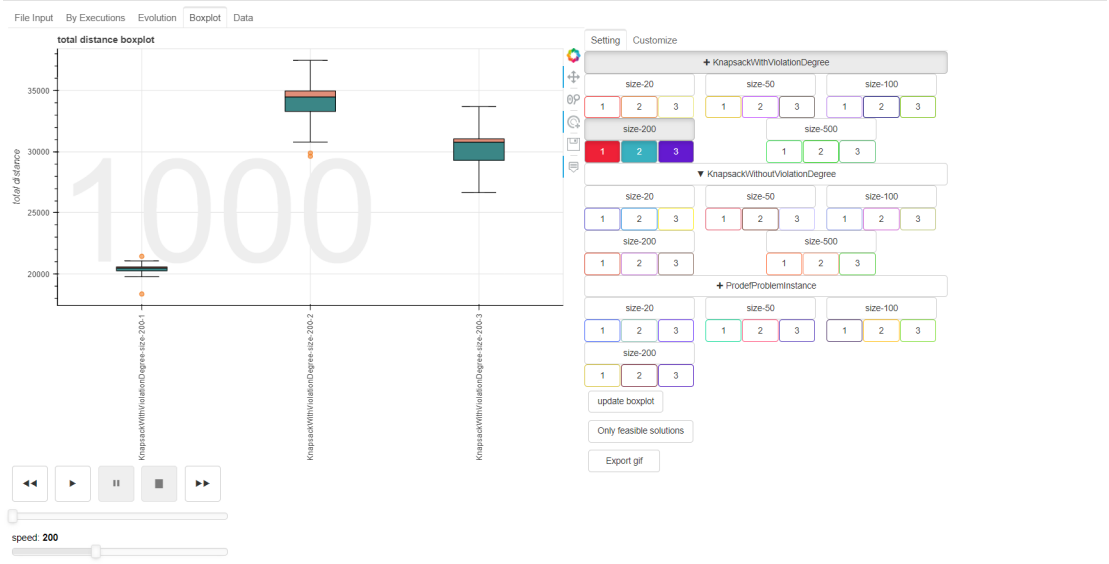
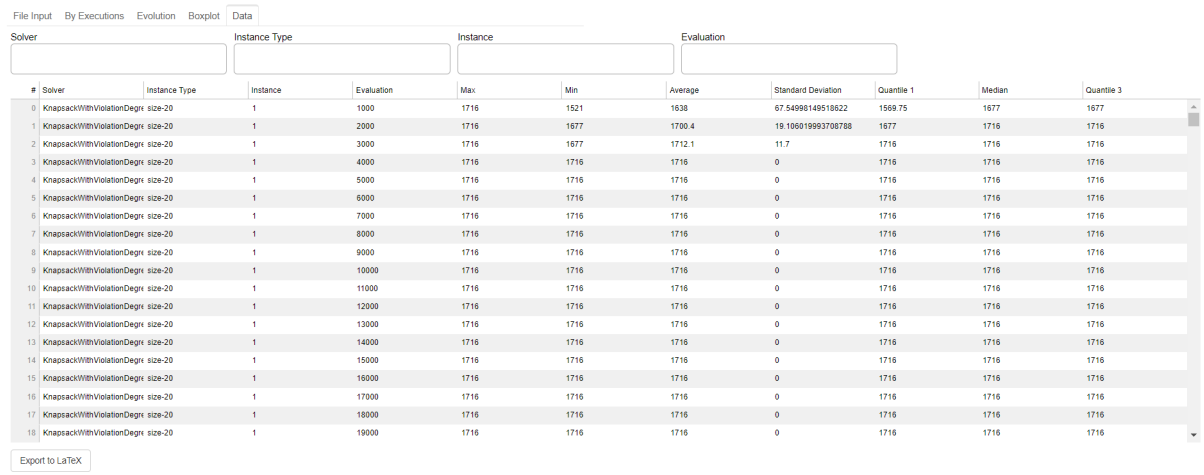
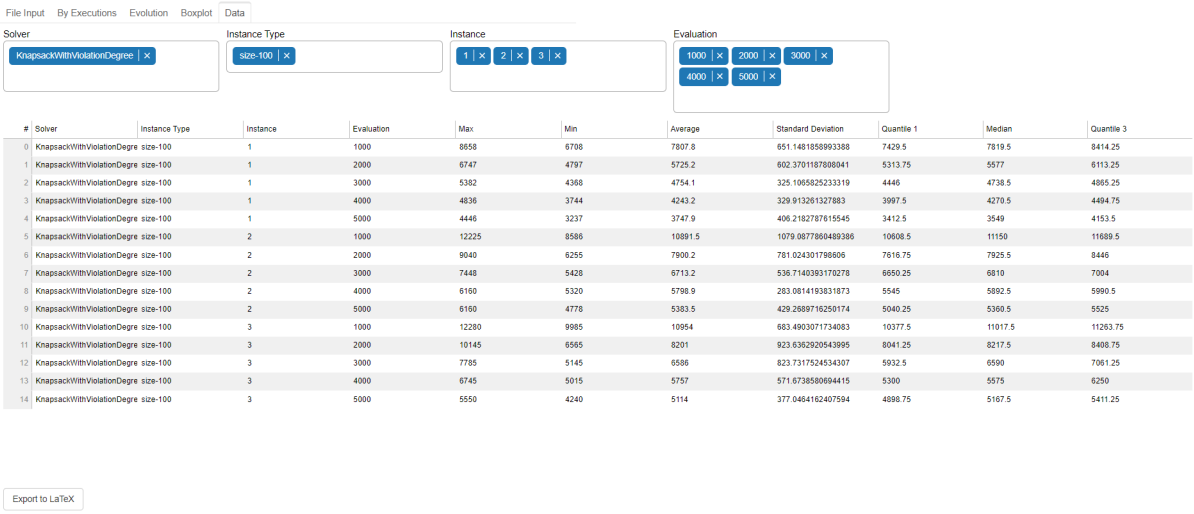


Figura 4.16: Vista del diagrama de cajas y bigotes reducido



#	Solver	Instance Type	Instance	Evaluation	Max	Min	Average	Standard Deviation	Quantile 1	Median	Quantile 3
0	KnapsackWithViolationDegre	size-20	1	1000	1716	1521	1638	67.5499814951862	1569.75	1677	1677
1	KnapsackWithViolationDegre	size-20	1	2000	1716	1677	1700.4	19.106019993708788	1677	1716	1716
2	KnapsackWithViolationDegre	size-20	1	3000	1716	1677	1712.1	11.7	1716	1716	1716
3	KnapsackWithViolationDegre	size-20	1	4000	1716	1716	1716	0	1716	1716	1716
4	KnapsackWithViolationDegre	size-20	1	5000	1716	1716	1716	0	1716	1716	1716
5	KnapsackWithViolationDegre	size-20	1	6000	1716	1716	1716	0	1716	1716	1716
6	KnapsackWithViolationDegre	size-20	1	7000	1716	1716	1716	0	1716	1716	1716
7	KnapsackWithViolationDegre	size-20	1	8000	1716	1716	1716	0	1716	1716	1716
8	KnapsackWithViolationDegre	size-20	1	9000	1716	1716	1716	0	1716	1716	1716
9	KnapsackWithViolationDegre	size-20	1	10000	1716	1716	1716	0	1716	1716	1716
10	KnapsackWithViolationDegre	size-20	1	11000	1716	1716	1716	0	1716	1716	1716
11	KnapsackWithViolationDegre	size-20	1	12000	1716	1716	1716	0	1716	1716	1716
12	KnapsackWithViolationDegre	size-20	1	13000	1716	1716	1716	0	1716	1716	1716
13	KnapsackWithViolationDegre	size-20	1	14000	1716	1716	1716	0	1716	1716	1716
14	KnapsackWithViolationDegre	size-20	1	15000	1716	1716	1716	0	1716	1716	1716
15	KnapsackWithViolationDegre	size-20	1	16000	1716	1716	1716	0	1716	1716	1716
16	KnapsackWithViolationDegre	size-20	1	17000	1716	1716	1716	0	1716	1716	1716
17	KnapsackWithViolationDegre	size-20	1	18000	1716	1716	1716	0	1716	1716	1716
18	KnapsackWithViolationDegre	size-20	1	19000	1716	1716	1716	0	1716	1716	1716

Figura 4.17: Vista de la tabla de datos sin filtrar



#	Solver	Instance Type	Instance	Evaluation	Max	Min	Average	Standard Deviation	Quantile 1	Median	Quantile 3
0	KnapsackWithViolationDegre	size-100	1	1000	8658	6708	7807.8	651.1481858993388	7429.5	7819.5	8414.25
1	KnapsackWithViolationDegre	size-100	1	2000	6747	4797	5725.2	602.3701187808041	5313.75	5577	6113.25
2	KnapsackWithViolationDegre	size-100	1	3000	5382	4368	4754.1	325.1065825233319	4446	4738.5	4865.25
3	KnapsackWithViolationDegre	size-100	1	4000	4836	3744	4243.2	329.91326132788 3	3997.5	4270.5	4494.75
4	KnapsackWithViolationDegre	size-100	1	5000	4446	3237	3747.9	406.2182787615545	3412.5	3549	4153.5
5	KnapsackWithViolationDegre	size-100	2	1000	12225	8586	10891.5	1079.0877860489386	10608.5	11150	11689.5
6	KnapsackWithViolationDegre	size-100	2	2000	9040	6255	7900.2	781.024301798606	7616.75	7925.5	8446
7	KnapsackWithViolationDegre	size-100	2	3000	7448	5428	6713.2	536.7140393170278	6650.25	6810	7004
8	KnapsackWithViolationDegre	size-100	2	4000	6160	5320	5798.9	283.0814193831873	5545	5892.5	5990.5
9	KnapsackWithViolationDegre	size-100	2	5000	6160	4778	5383.5	429.2689716250174	5040.25	5360.5	5525
10	KnapsackWithViolationDegre	size-100	3	1000	12280	9985	10954	683.4903071734083	10377.5	11017.5	11263.75
11	KnapsackWithViolationDegre	size-100	3	2000	10145	6565	8201	923.6362920543995	8041.25	8217.5	8408.75
12	KnapsackWithViolationDegre	size-100	3	3000	7785	5145	6586	823.7317524534307	5932.5	6590	7061.25
13	KnapsackWithViolationDegre	size-100	3	4000	6745	5015	5757	571.6738580694415	5300	5575	6250
14	KnapsackWithViolationDegre	size-100	3	5000	5550	4240	5114	377.0464162407594	4898.75	5167.5	5411.25

Figura 4.18: Vista de la tabla de datos filtrada

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

Empezando con una etapa de investigación sobre $jMetal$ y $Prodef$ ambos herramientas desarrolladas para la facilitación del diseño, implementación y evaluación de algoritmos basados en metaheurísticas para problemas de optimización combinatoria para conseguir un conjunto de soluciones que representar. Este análisis supuso un reto a la hora de la instalación y configuración de un estudio experimental con la herramienta $jMetal$ debido a la falta de conocimientos en el tema y en el lenguaje $Java$.

A continuación, se realizó una fase de análisis de librerías de visualización de datos interactivos y creación de cuadros de mando para la generación de gráficos interactivos, diagramas y tablas de manera dinámica. Se analizaron las librerías $Bokeh$, $Panel$ y $Dash$ realizando una serie de pruebas con gráficos básicos y probando la capacidad de estas para realizar visualizaciones interactivas.Se concluyó que para el caso de uso la mejor opción era $Bokeh$ debido a su mayor flexibilidad para la realización de visualizaciones complejas y su sencillez en el momento de utilizarlo.

En la siguiente fase, se comenzó con el diseño de la interfaz empezando por implementar un gráfico de ejecuciones mediante el que a través de las sucesivas evaluaciones de los algoritmos se representaran la soluciones con los valores que toma la función objetivo(representado en el eje y) en el número de ejecuciones(representadas en el eje x) a lo largo del tiempo. Para poder visualizar esa evolución a lo largo del tiempo se desarrolló un reproductor que como si se tratara de un reproductor de vídeos controlara el movimiento del gráfico.

Lo siguiente fue crear los botones de personalización de la interfaz con los que se pudiera controlar lo que se veía o no en la zona del gráfico. Añadiendo también un apartado para cambiar la forma de los marcadores, el color, y los títulos de los ejes y del propio gráfico.

Adicionalmente se automatizó la manera de cargar los datos de las soluciones mediante la subida de un fichero $JSON$ que contiene la estructura del estudio experimental y las soluciones para crear los $dataframes$ de toda la estructura de la aplicación de manera automática.

También se implementaron las representaciones para los gráficos de estadísticas, calculando el mínimo, máximo y promedio, y para los diagramas de cajas. Definiendo para ellos sus respectivas opciones de personalización.

Finalmente se implementaron las exportaciones a gif de las animaciones desarrolladas, así como una última vista con una tabla de datos que es exportable a LaTeX mediante un botón y a la que se le puede aplicar filtros.

Durante el desarrollo del proyecto me he enfrentado a diferentes dificultades entre las que se destacan la curva de aprendizaje a la hora de tratar con los resolutores de problemas combinatorios basados en metaheurísticas y la integración de todas las partes de interfaz para que funcionen en conjunto.

Como conclusión final, se ha desarrollado una interfaz gráfica que permite la evaluación de la calidad de las soluciones de los problemas de optimización combinatoria en la dimensión de las funciones objetivo, proporcionando una serie de gráficos interactivos que pueden ayudar en el estudio de viabilidad de los algoritmos basados en metaheurísticas utilizados y que permiten la exportación de las gráficas y estadísticas obtenidas.

5.2. Líneas Futuras

A continuación se detallan una serie de futuras líneas de trabajo para continuar con la implementación:

- **Implementación de una herramienta para la visualización interactiva de soluciones en la dimensión de las variables:** Desarrollo de una serie de gráficas para la representación visual de las variables permitiendo la interacción con lo gráficos para explorar las diferentes combinaciones de valores de la variable y como estas afectan a la calidad de las soluciones
- **Mejora de la interfaz gráfica con el uso de más interacciones con el usuario y un mejor diseño de interfaz:**Añadiendo a la interfaz gráfica más interacciones con el usuario haciendo uso de animaciones y notificaciones para la pantallas de carga, los botones de exportación. Además de incluir el uso de plantillas $html$ para una mejor distribución de los elementos de la interfaz.
- **Optimización de la interfaz para la reducción de los tiempos de carga de los ficheros de configuración:** Refactorización de las funciones de carga de la interfaz, optimizando el código para la reducción de los tiempos de carga de estos.

Capítulo 6

Summary and Conclusions

Starting with a research stage on $jMetal$ and $Prodef$, both tools developed to facilitate the design, implementation, and evaluation of metaheuristic-based algorithms for combinatorial optimization problems to obtain a set of solutions.

Next, an analysis phase of interactive data visualization libraries and dashboard creation for generating dynamic interactive charts, diagrams, and tables was conducted. The libraries $Bokeh$, $Panel$, and $Dash$ were examined, and ultimately this project was implemented using $Bokeh$.

In the next phase, the interface design began by implementing a execution plot. Through successive algorithm evaluations, the solutions were represented with their corresponding objective function values (on the y-axis) against the number of executions (on the x-axis) over time. To visualize this evolution over time, a player was developed, which controlled the movement of the plot.

Next, customization buttons were created for the interface to control the visibility of elements in the chart area. An option was added to change the shape of the markers, their color, and the titles of the axes and the plot itself.

Representation for statistical charts was also implemented, calculating the minimum, maximum, and average values, as well as box plots. Customization options were defined for these representations.

Finally, exports to GIF format for the developed animations were implemented. A last view was added, which included a data table that could be exported to LaTeX using a button and had filter capabilities and the process of loading solution data was automated by uploading a JSON file containing the experimental study structure and the solutions. .

In conclusion, a graphic interface has been developed that enables the evaluation of solution quality for combinatorial optimization problems in terms of objective functions. It provides a set of interactive charts that can assist in the feasibility study of the utilized metaheuristic algorithms. The interface also allows for the export of obtained graphs and statistics.

Capítulo 7

Presupuesto

En este capítulo se detalla el presupuesto para el trabajo final de grado. Como podemos ver se superan las 300 horas correspondientes a los 12 créditos ECT de la asignatura.

Descripción	Horas	Precio/Horas	Precio
Instalación y configuración de jMetal	48	9.5	456
Análisis de librerías de visualización	48	9.5	456
Diseño de la interfaz gráfica	35	12	420
Desarrollo de la interfaz gráfica	180	13	2340
Documentación	20	9.5	190
Total	331	-	3862.00

Tabla 7.1: Presupuesto

Bibliografía

[1] García Pérez, Andrés Calimero, *Prodef: meta-modelado de problemas de optimización combinatoria*, Universidad de La Laguna, 2020. URL: http://riull.ull.es/xmlui/handle/915/20601

[2] Ordoñez Morales, Miguel, *Prodef: diseño, implementación y experimentación con nuevos resolutores*, Universidad de La Laguna, 2022. URL: http://riull.ull.es/xmlui/handle/915/26852

[3] Antonio J. Nebro, *jMetal project documentation*, URL: https://jmetal.readthedocs.io/en/latest/, visitado 09-04-2023

[4] Bokeh, *Bokeh reference guide*, URL: https://docs.bokeh.org/en/latest/docs/reference.html, visitado 24-09-2022

[5] Dash by Plotly, *Dash Python User Guide*, URL: https://docs.bokeh.org/en/latest/docs/reference.html, visitado 21-05-2022

[6] Holoviz contributors., *Panel*, URL: https://panel.holoviz.org/api/index.html, visitado 21-05-2022

[7] Hans Rosling's, *Gapminder*, URL: https://demo.bokeh.org/gapminder, visitado 24-09-2022