Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Prodef-Solution: Interfaz para la representación y visualización de soluciones

Prodef-Solution: Interface for the representation and visualization of solutions

Yeixon Reinaldo Morales Gonzalez

La Laguna, 13 de septiembre de 2022

Da. **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. **Daniel González Expósito**, con N.I.F. 42.242.696-E, graduado en Ingeniería Informática por la Universidad de La Laguna, como cotutor

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

Que la presente memoria titulada:

”*Prodef-Solution: Interfaz para la representación y visualización de soluciones*”

ha sido realizada bajo su dirección por D. **Yeixon Reinaldo Morales Gonzalez**, con N.I.F. 51.209.415-F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de septiembre de 2022

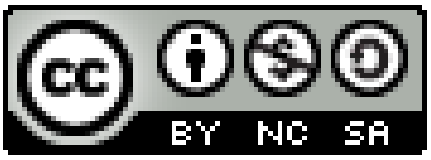
Agradecimientos

Para empezar quiero darle las gracias a mi tutora, Gara Miranda, quien me ha apoyado y guiado a lo largo de este trabajo, junto al co-tutor Daniel González, que sin ellos este trabajo no hubiese sido posible.

También quiero agradecer a Alberto Cabrera, el tutor externo de la asignatura de Practicas Externas por en el período de prácticas haberse tomado la molestia de resolverme dudas relacionadas a este proyecto.

Por último, quiero agradecer a todos mis compañeros y a mi familia, por apoyarme cuando mis ánimos estaba en el piso. En especial, quiero hacer mención de mis padres, que siempre estuvieron ahí motivándome para así sacar este proyecto adelante.

Licencia



Resumen

En la actualidad los algoritmos de optimización son utilizados en diferentes campos de la investigación. Sin embargo su uso en los campos empresariales es reducido, ya que no es sencillo introducirlo en según qué entorno por la complejidad que conlleva. Y es así como surge Prodef que tiene como objetivo ir un paso más allá y proporcionar una herramienta de más alto nivel, a través de la cual cualquier usuario pueda modelar problemas de optimización y ejecutarlos sin necesidad de tener conocimientos previos sobre algoritmos evolutivos ni sobre programación. A pesar de que existen algunos frameworks para optimización con este tipo de meta-heurísticas, su utilización requiere de un cierto nivel de conocimiento sobre la técnica a emplear y también sobre el lenguaje de programación utilizado por el propio framework. La estructura general de Prodef se desarrolló en unos Trabajos de Fin de Grado previos, gracias al cual se dispone de una API en la que se puede solicitar ejecutar problemas a través de un JSON, especificando el modelo formal del problema a través de un lenguaje propio denominado ProdefLang, todo esto se hace de forma gráfica a través de una interfaz ya existente, que permite al usuario modelar sus problemas haciendo uso de bloques, que conectándolos de ciertas formas, se puede plantear un problema, dándole la facilidad al usuario de abstraerse del código que hay por detrás.

En el presente trabajo se ha logrado desarrollar una interfaz gráfica, Prodef-Solution. Gracias a esta, los usuarios que utilicen la herramienta podrán visualizar los resultados de una forma más representativa. De esta forma se hace más fácil la comprensión de los resultados para aquellos usuarios no expertos. Las representaciones de las soluciones se han agrupado basándonos en el tipo de solución, de las cuales algunas pueden ser un vector de booleano, una matriz de booleanos, un vector de enteros con permutación, una matriz de enteros con permutación y una matriz de enteros, permitiendo así agrupar diferentes problemas para poder darles una representación genérica.

Como resultado de este Trabajo de Fin de Grado se ha logrado crear una interfaz web capaz de representar los resultados de diferentes problemas adaptándose a los distintos grupos, permitiendo así poder llevar a cabo una representación genérica, que por un lado facilita las representaciones de los diferentes problemas tomando los puntos comunes entre algunos de ellos, mientras que por otro lado tenemos el inconveniente de que no se puede hacer 100 % representativa a cada problema, ya que esto le aportaría una gran complejidad, debido a que tendríamos que tener definida cada representación para cada problema que exista y eso seria muy ineficiente.

Palabras clave: Optimización combinatoria, Visualización de resultados, Algoritmos evolutivos, Herramienta web, Prodef-ULL

Abstract

Currently optimization algorithms are used in different fields of research. However, its use in business fields is limited, since it is not easy to introduce it in a certain environment due to the complexity it entails. And this is how Prodef was created, which aims to go one step further and provide a higher level tool, through which any user can model optimization problems and execute them without the need for prior knowledge of evolutionary algorithms or programming. Although there are some frameworks for optimization with this type of meta-heuristics, their use requires a certain level of knowledge about the technique to be used and also about the programming language used by the framework itself. The general structure of Prodef was developed in previous Final Degree Projects, thanks to which an API is available in which problems can be requested to be executed through a JSON, specifying the formal model of the problem through its own language called ProdefLang, all this is done graphically through an already existing interface, which allows the user to model their problems using blocks, which by connecting them in certain ways, can pose a problem, giving the user the facility to abstract from the code behind.

In the present work it has been possible to develop a graphical interface, Prodef-Solution. Thanks to this, users who use the tool will be able to visualize the results in a more representative way. This makes it easier for non-expert users to understand the results. The representations of the solutions have been grouped based on the type of solution, of which some can be a boolean vector, a boolean matrix, a permutation integer vector, a permutation integer matrix and an integer matrix, thus allowing different problems to be grouped in order to give them a generic representation.

As a result of this Final Degree Project, it has been achieved to create a web interface capable of representing the results of different problems adapting to the different groups, thus allowing a generic representation to be carried out, that on one side facilitates the representations of the different problems taking the common points between some of them, while on the other side we have the inconvenience that each problem cannot be 100 % representative, since this would add great complexity, since we would have to have each representation defined for every problem that exists and that would be very inefficient.

Keywords: Combinatorial optimization, Visualization of results, Evolutionary algorithms, Web tool, Prodef-ULL

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

El diseño y la experimentación con algoritmos evolutivos y otros tipos de meta-heurísticas bio-inspiradas es un campo de investigación muy amplio y activo en la actualidad. Se ha demostrado que este tipo de técnicas son capaces, en general, de ofrecer soluciones de alta calidad para problemas de optimización complejos y que, difícilmente, serían abordables mediante técnicas exactas. Si un usuario sin conocimientos específicos sobre meta-heurísticas y/o programación desea aplicar este tipo de técnicas para resolver un problema concreto (sobre el cual sí que tiene conocimiento y/o experiencia profunda) podrá evitar implementar desde cero este tipo de algoritmos. Esto se debe a que en la actualidad existen algunas herramientas que proporcionan esqueletos o librerías para optimización de problemas basándonos en técnicas meta-heurísticas preestablecidas. Estas herramientas suponen una gran ventaja respecto a la implementación directa del algoritmo desde el principio, no solo en términos de reutilización de código sino también en cuanto a metodología y claridad de conceptos.

El problema de estas herramientas es que la definición del problema se adapta al tipo de técnica a aplicar, con lo cual, si el usuario desea resolver el problema mediante la aplicación de técnicas diferentes, tendrá que definir el problema de formas distintas o bien implementar funciones adaptadas a cada una de las estrategias a utilizar. Para ello, es necesario que el usuario entienda el diseño interno de la herramienta que vaya a utilizar, pues esto le permitirá seleccionar el tipo de técnicas, operadores y resto de configuración adecuada para el problema que necesita resolver. Para todo esto se presupone además que el usuario tiene conocimientos sólidos sobre programación. Por lo tanto, debido a la complejidad intrínseca de estos algoritmos y también a sus particularidades a la hora de implementarlos y adaptarlos a cada uno de los problemas, no se ha conseguido extender su uso fuera del ámbito de la investigación convencional.

Dada estas circunstancias, se plantea la necesidad de diseñar e implementar nuevas herramientas que ayuden a mejorar la aplicabilidad de estas técnicas de optimización. El gran reto es que no podemos diseñar una herramienta más a añadir a la lista de ya existentes, sino que debemos ser capaces de proporcionar una clara separación entre la definición del problema y la determinación del método de optimización a aplicar. También es necesario dar un salto cualitativo en lo que respecta a sencillez de uso, tratando así de dejar atrás todo lo que implica complejas implementaciones o configuraciones a bajo nivel.

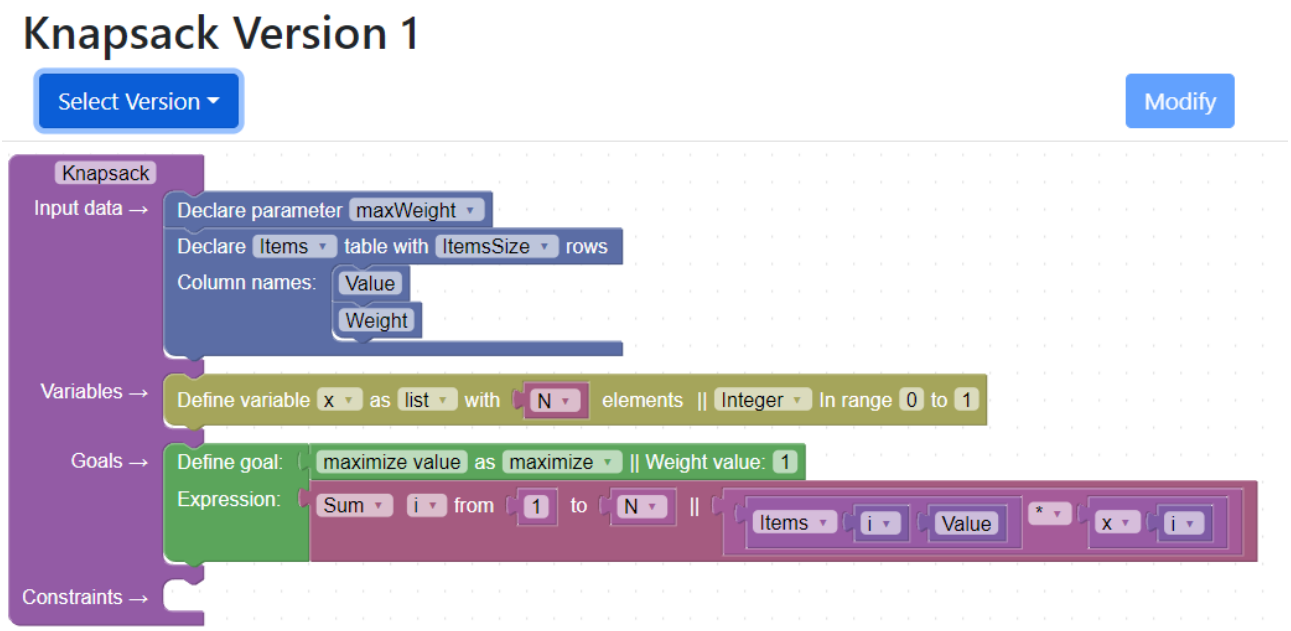


Figura 1.1: Interfaz para crear/actualizar el diseño de un problema

1.1. Antecedentes y estado actual del tema

En este contexto surge Prodef, una herramienta que nace con el objetivo de extender el uso de estas técnicas meta-heurísticas fuera del ámbito de la investigación convencional, tratando de acercar incluso el uso de estas técnicas a pequeñas y medianas empresas. De esta forma, los usuarios pueden aplicar técnicas meta-heurísticas para la resolución de sus problemas obteniendo, en la medida de lo posible, soluciones razonables para sus problemas. Todo esto aislando al usuario de la implementación interna y de conceptos relacionados con los algoritmos evolutivos.

La actual versión de Prodef es fruto de varios Trabajos de Fin de Grado, de los cuales estos dos son los que conforman el esqueleto principal de la herramienta:

- Prodef: meta-modelado de problemas de optimización combinatoria, 2020. Andrés Calimero García Pérez[1].
- Prodef-GUI: Interfaz gráfica para el modelado de problemas, 2021. Daniel González Expósito[2].

Prodef se basa en un meta-modelo que permite abstraer las características esenciales de un problema de optimización combinatoria. A partir de este modelo, se genera una traducción directa a una herramienta externa de optimización basada en metaheurísticas bio-inspiradas. A la vez que esta herramienta, se diseñó un DSL llamado ProdefLang, que se creó con el fin de poder especificar la función objetivo, diferentes variables y restricciones de forma sencilla. Este lenguaje combina la formulación matemática con el paradigma orientado a objetos. Inicialmente, la especificación de problemas en este lenguaje se hacía a través de un fichero JSON, pero una de las últimas nuevas funciones de Prodef es la posibilidad de especificar las características del problema gráficamente utilizando bloques (Figura 1.1), favoreciendo aún más la experiencia de usuario.

Por lo tanto, actualmente Prodef permite trabajar con una interfaz gráfica (desarrollada por Daniel González [2]) que permite al usuario modelar su problema haciendo uso de

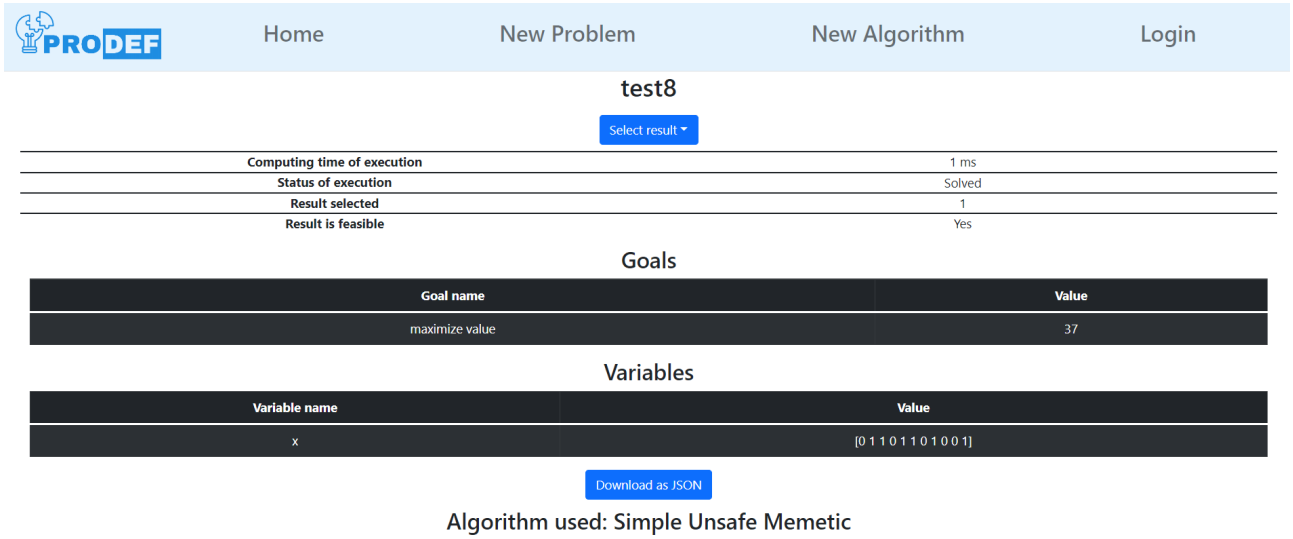


Figura 1.2: Interfaz para la visualización de resultados

un conjunto de bloques – desarrollados haciendo uso de la librería Blockly - y crear varias versiones del mismo. A su vez, cada versión de un problema puede ser utilizada o probada con diferentes datos (a los cuales se les denomina instancias). Todos estos datos se estructuran, por separado, en un formato JSON, los cuales son almacenados en una base de datos MongoDB. Cuando el usuario quiera resolver el problema con una versión e instancia en concreto, entonces la herramienta toma los datos almacenados y los combina en un JSON con toda la información. Este fichero de especificación conjunta de problema-instancia se envía a un servidor (desarrollado por Andrés Calimero García [1]), donde se pondrá en marcha un resolutor que resuelva el problema. En este proceso se devuelve un identificador del problema y una contraseña, la cual se utilizará a posteriori para obtener el estado del problema. Una vez finalizada la ejecución del resolutor (que en realidad simula el comportamiento de un algoritmo evolutivo básico) la mejor solución encontrada durante el proceso de búsqueda se mostrará al usuario de una forma muy primitiva, de difícil comprensión para un usuario no experto, o bajos conocimientos en el entorno (Figura 1.2). El objetivo de este proyecto será el diseño e implementación de una interfaz que permita visualizar, analizar y comparar, de una forma intuitiva, diferentes soluciones a un determinado problema.

1.2. Objetivos

Tal y como se puede apreciar en la figura anterior (Figura 1.2), en este momento la plataforma Prodef, muestra sus datos de una forma muy primitiva (vector de variables con los valores correspondientes), lo cual puede generar confusión o poca información para usuarios no expertos. Lo que se pretende realizar en este Trabajo de Fin de Grado es plantear y analizar diferentes alternativas para la especificación y visualización de las soluciones encontradas a cada problema. El objetivo es proporcionar a los usuarios una interfaz en la que los resultados puedan ser fácilmente interpretados y/o comparados. Además, sería también deseable que la interfaz permitiera, en la medida de lo posible, interactuar con las soluciones encontradas.

A modo de ejemplo a continuación se incluyen unas imágenes que representan soluciones auto-explicativas a problemas como la planificación de recursos (Figura 1.3) [9]

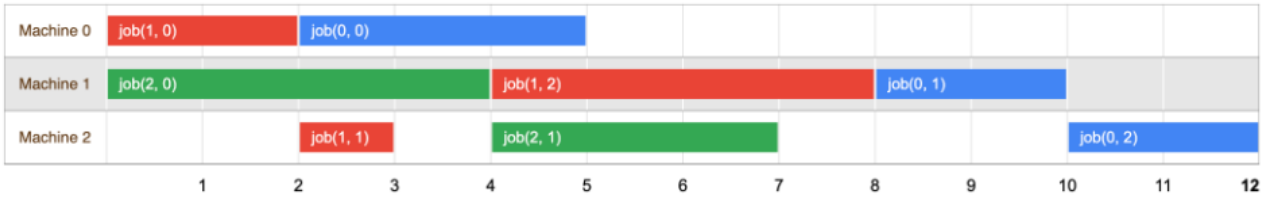


Figura 1.3: Problema de planificación de recursos

Figura 1.4: Problema de la mochila

y el problema de la mochila (Figura 1.4) [5]. Ambas representaciones obtenidas de la herramienta Google OrTools [4].

El problema de la mochila tiene como objetivo conseguir meter la mayor cantidad de ítems dentro de un contenedor tales que no sobrepasen el espacio del contenedor ni el peso máximo del mismo.

En la imagen (Figura 1.4) se aprecia un conjunto de círculos que serian los ítems, los cuales tienen un valor, que podría ser su peso o tamaño; por otro lado tenemos una caja que sería nuestro contenedor, donde se puede apreciar que hay círculos dentro, pues estos círculos serian los ítems seleccionados de forma que optimiza el espacio y el peso. Más adelante se define de forma formal este problema.

Esta imagen está animada, mostrando como se meten los ítems dentro del contenedor [5].

El objetivo del problema planificación de recursos es conseguir asignar una tarea a una máquina, en un tiempo óptimo, consiguiendo una eficacia a la hora de realizar las tareas [9].

En la imagen (Figura 1.3) se puede ver como las máquinas realizan ciertas tareas a lo largo de un tiempo. Ejemplo de la máquina 1, que realiza la tarea 1 durante 2 unidades de tiempo y seguido la tarea 0 durante 3 unidades de tiempo. Más adelante se define de forma formal este problema.

1.3. Actividades a realizar

Con el fin de diseñar e implementar esta interfaz gráfica para la visualización e interpretación de soluciones, será necesario abordar las tareas o actividades que se relacionan a continuación:

- **Análisis de Prodef y de ProdefLang**: El objetivo de esta tarea es abordar el estudio de la herramienta Prodef, comprendiendo su estructura modular y su funcionamiento interno. También será fundamental comprender la sintaxis de ProdefLang, lenguaje que se ha definido para el modelado de los problemas. Esto se cumple en el capítulo 2.

- **Análisis de problemas y sus soluciones**: Analizar diferentes problemas (clásicos o estándar) de optimización combinatoria, realizando su modelado en Prodef y recogiendo las principales características de las soluciones para cada tipo/familia de problema. Esto se cumple en el capítulo 3.

- **Diseño e implementación de una interfaz gráfica para la representación de las soluciones**: Analizar posibles formatos en los que especificar las soluciones obtenidas. Teniendo en cuenta dicho formato, diseñar una interfaz que permita visualizar y conceptualizar el significado de cada solución en el contexto del problema. Esto se cumple en el capítulo 4.

- **Validación y análisis de casos**: Realizar las pruebas necesarias para comprobar en qué medida la interfaz se adapta a diferentes problemas y/o características de soluciones. Esto se cumple en el capítulo 5.

- **Documentación**: Elaboración del material de soporte para el uso de la herramienta así como la memoria del TFG y demás documentación requerida durante la realización de la asignatura de TFG.

Capítulo 2

Prodef ULL

En el capítulo anterior se ha hablado acerca del estado actual de Prodef, en este capítulo trataremos más en detalle su implementación y diferentes módulos por los que está compuesto.

2.1. Módulos

Prodef está compuesto por uno grupo de módulos desarrollados en TypeScript, que se relacionan entre ellos. Estos módulos se pueden separar en dos basados en su importancia, por un lado los “cores” que son los fijos, que cuestan mucho remplazarlos, ya que son la base que permite expandir nuevos módulos; por otro lado están los módulos que expanden el proyecto, que son fáciles de modificar y/o remplazar.

- **Prodef GUI - Frontend:** Una interfaz gráfica que permite modelar problemas, a través del uso de la librería Blockly; crear instancias, ejecución de problemas, visualización de soluciones, que como se mencionó anteriormente este era muy primitivo.
- **Prodef GUI - Backend:** Este módulo se encarga de almacenar las soluciones de problemas ya antes solucionados, esta se comunica con Prodef GUI- Frontend.
- **Prodef Compiler:** Módulo base que permite compilar todo el programa para poder resolver, desde el cual se pueden sacar otros compiladores desarrollados en otros lenguajes de programación a partir de este.
- **Prodef Solver:** Módulo que resuelve el problema que recibe en el lenguaje de ProdefLang. Al igual que el compilador se puede crear uno para diferentes lenguajes.

2.2. Funcionamiento

Para comenzar partimos de esta imagen (Figura 2.1), en la cual podemos apreciar en la parte superior el botón para crear un nuevo problema, una vez le pulsamos, se nos abre una interfaz, en la que a través de la librería Blockly [3], podemos definir un problema, ejemplo de esto esta en la Figura 1.1. En la que se aprecia como se definen:

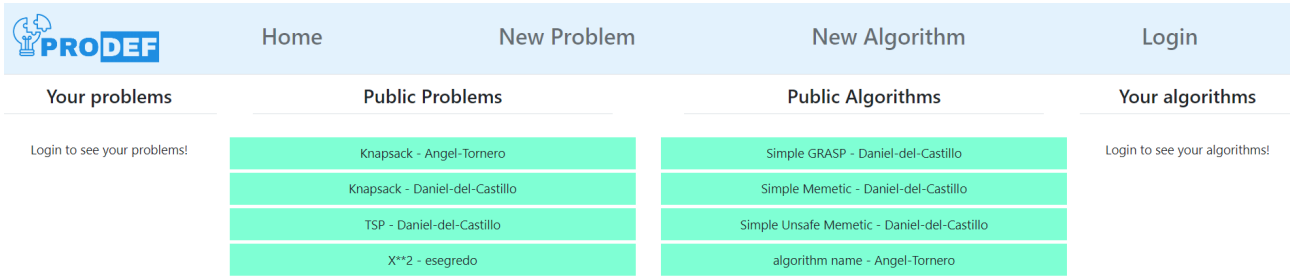


Figura 2.1: Interfaz de selección de problema

- **Datos de entrada**: Permite como entrada parámetros o tablas, en caso de querer definir vectores podemos simplemente utilizar una tabla con una única columna. En la figura 1.1 se aprecia como se esta declarando un parámetro que indica el peso máximo que permitirá la mochila y seguido se define una tabla la cuál contendrá el valor y peso de cada ítem.
- **Variables**: Incógnitas del problema cuyo valor debemos obtener para hallar una solución. El tipo puede ser: `number`, `list`, `permutationlist` o `matrix`. En la figura 1.1 de ejemplo se define una variable de tipo lista de enteros entre 0 y 1 lo cual lo convierte en una variable booleana.
- **Objetivos**: Funciones objetivo a maximizar o minimizar. Como se ve, en la figura 1.1 se indica la necesidad de maximizar la función objetivo, seguido esta la expresión que hay que maximizar, que seria el sumatorio de i en pesando en 1 hasta los N ítems.
- **Restricciones**: Condiciones que deben cumplir las soluciones para ser válidas. En el ejemplo no se pueden apreciar, pero en este apartado al igual que con las funciones objetivo se añaden expresiones a modo de restricción.

Una vez definido todo el problema, este se transforma en un JSON.

```
{
   "name": "Basic binary knapsack problem",
   "description": "Optional description (optimal: 130)",
   "variables": [
     {
       "name": "Items in the knapsack",
       "shape": {
         "type": "vector",
         "isPermutation": false,
         "size": {
           "fixed": false,
           "value": "N"
         }
       },
       "symbol": "x",
       "within": "integers",
       "range": {
         "lowerBound": 0,
         "upperBound": 1
       }
     }
```

```
    ],
    "goals": [
      {
        "name": "Maximize the value of the items",
        "expression": "sum x[i]*item[i].value over i=(1:N)",
        "sense": "maximize",
        "weight": 1
      }
    ],
    "constraints": [
      {
        "expression": "sum x[i]*item[i].weight over i=(1:N) <= MaxWeight",
        "name": "Limit the total weight of the items in the knapsack"
      }
    ]
  }
```

En este JSON se aprecia el nombre que se le dio al problema, con una descripción, seguido de los parámetros definidos en la interfaz, que estos serían los datos de entrada; las variables que contenderán el resultado del problema donde se indican de que tipo son, si son con permutación y el rango del valor permitido y por último encontramos la función objetivo y las restricciones. Este JSON se almacena en una Base de datos de MongoDB (ver esquema 2.3), la cual se usa para poder almacenar los diferentes problemas que se definan y sus distintas versiones.

Ahora tendremos que definir lo que son las instancias, que no son más que los valores que toman las variables de entrada al problema.

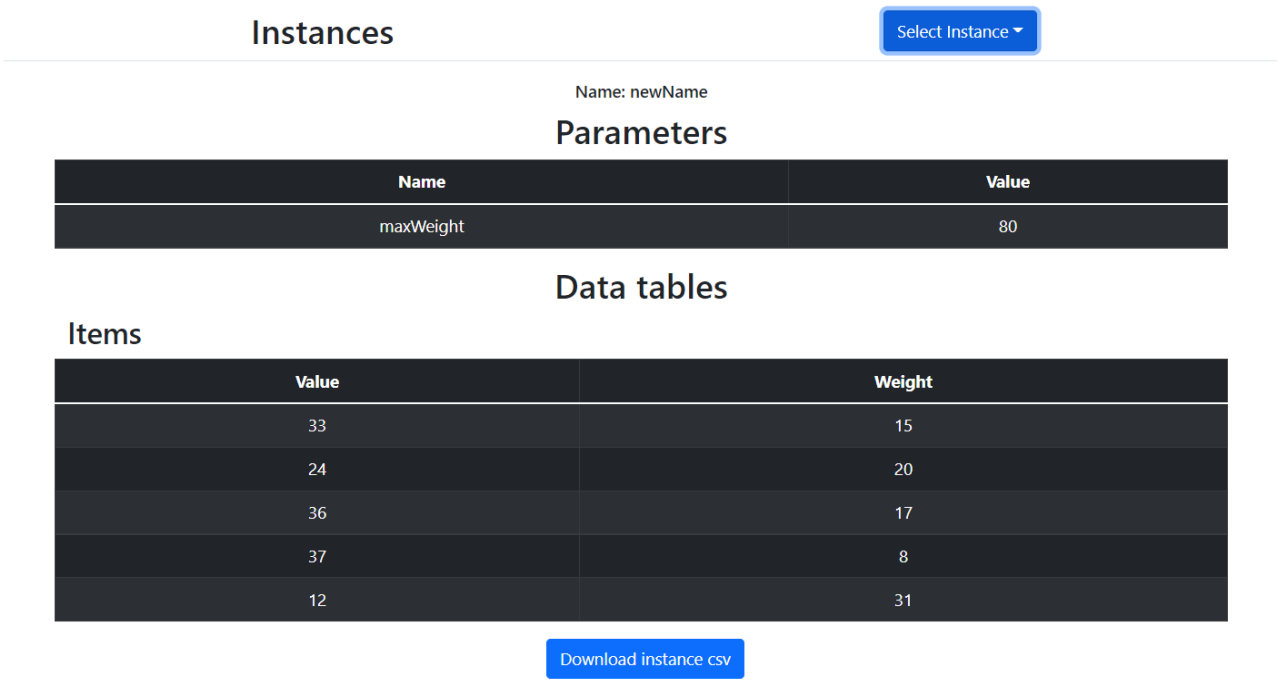


Figura 2.2: Interfaz para la creación de instancias

Tomando como referencia esta imagen (Figura 2.2), se está creando una instancia para el problema de la mochila, donde se aprecia en la parte superior de la imagen los parámetros definidos, en este caso el peso máximo que permite la mochila y más abajo

como una tabla, el valor y el peso de los ítems en cada fila de la tabla. Toda esta instancia se transforma en otro JSON.

```
{
   "parameters": [
     {
       "symbol": "N",
       "value": 5,
       "name": "Number of items"
     },
     {
       "symbol": "MaxWeight",
       "value": 80,
       "name": "Maximum weight"
     }
   ],
   "objects": [
     {
       "attributes": [
         {
           "attribute": "name",
           "value": "Item 1"
         },
         {
           "attribute": "value",
           "value": 33
         },
         {
           "attribute": "weight",
           "value": 15
         }
       ],
       "class": "item"
     },
     {
       "attributes": [
         {
           "attribute": "name",
           "value": "Item 2"
         },
         {
           "attribute": "value",
           "value": 24
         },
         {
           "attribute": "weight",
           "value": 20
         }
       ],
       "class": "item"
     },
     {
       "attributes": [
         {
           "attribute": "name",
           "value": "Item 3"
         },
```

```
            {
              "attribute": "value",
              "value": 36
            },
            {
              "attribute": "weight",
              "value": 17
            }
          ],
          "class": "item"
        },
        {
          "attributes": [
            {
              "attribute": "name",
              "value": "Item 4"
            },
            {
              "attribute": "value",
              "value": 37
            },
            {
              "attribute": "weight",
              "value": 8
            }
          ],
          "class": "item"
        },
        {
          "attributes": [
            {
              "attribute": "name",
              "value": "Item 5"
            },
            {
              "attribute": "value",
              "value": 12
            },
            {
              "attribute": "weight",
              "value": 31
            }
          ],
          "class": "item"
        }
      ]
    }
```


Dicho JSON se guarda separado del problema, en la misma base de datos (ver esquema 2.3) en la que se guardan las definiciones de los problemas, esto con el mismo objetivo de poder crear diferentes instancias para luego usarlas y poder así ejecutar varias versiones de un mismo problema, con varias instancias. Dando como resultado otro JSON que es la unión de ambos.

La unión de estos JSONs es lo que se le pasa al resolutor desarrollado por Andrés Calimero García [1]. Generando como resultado, un JSON que contienen una ID de

ejecución y una clave, que con estos datos se hace una petición a la API del resolutor 2.3, para así obtener el resultado del problema que esté sería otro JSON, que contiene toda la definición del problema y las instancias que vimos anteriormente, pero junto a este nuevo JSON se le añadió otros atributos como es, la solución al problema.

Para no abarcar tanto espacio se optó por mostrar los atributos añadidos al JSON. Para ver la versión completa diríjase al Apéndice A.1


```
{
    "state": "resolved",
    "solution": {
      "results": [
        {
          "isFeasible": false,
          "goalValues": [
            {
              "expression": "sum x[i]*item[i].value over i=(1:N)",
              "sense": "maximize",
              "value": 130,
              "name": "Maximize the value of the items",
              "weight": 1
            }
          ],
          "variableValues": [
            {
              "symbol": "x",
              "value": [0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1],
              "within": "integers",
              "name": "Items in the knapsack",
              "range": {
                "lowerBound": 0,
                "upperBound": 1
              },
              "shape": {
                "type": "vector",
                "isPermutation": false,
                "size": {
                  "fixed": false,
                  "value": "N"
                }
              }
            }
          ]
        },
      ],
      "computingTime": 5015
    },
    "stateMessage": "The problem was successfully solved",
    "problem": {...}
    "lastUpdate": 1605879754703
  }
```


En este JSON resultante, apreciamos un atributo, "`state`", que define si el problema esta resuelto o aun está pendiente, también podemos destacar que dentro de las soluciones, pueden haber varios resultados, que a su vez, cada resultado tiene sus atributos,

como son "`isFeasible`" que dice si el resultado es fiable; "`goalValues`" que es la función objetivo; "`computingTime`" el tiempo que tardo el resolutor en obtener un resultado y por último "`variableValues`"que es la variable con la respuesta del problema. En esta parte del JSON se puede ver reflejado el tipo de variable que es "`type`", el valor que tomo como solución óptima "`value`", cuál es el nombre de dicha variable "`name`", los rangos que puede tomar "`range`", si es una permutación "`isPermutation`", entre otros campos.

Este último JSON se representa actualmente en Prodef como se aprecia en la imagen (Figura 1.2), en la cual se ve para empezar el nombre de la ejecución, seguido un selector que permite cambiar entre los diferentes resultados. Seguido encontramos una tabla, en la cual se aprecian datos generales de la solución del problema, como son el tiempo de cómputo, el estado de la solución, el resultado seleccionado y si dicho resultado es fiable. Más abajo tenemos la representación de la variable que contienen la solución que como se aprecia en este problema es un vector de booleanos ([0 1 1 0 1 1 0 1 0 0 1]), el cual indica que los ítems de la posición 1, 2, 4, 5, 7, 10 son los que consiguen cumplir la función objetivo y las restricciones del problema, mientras que el resto de ítems no son seleccionados.

Aquí tenemos un esquema 2.3 de la estructura que hay detrás de este complejo sistema.

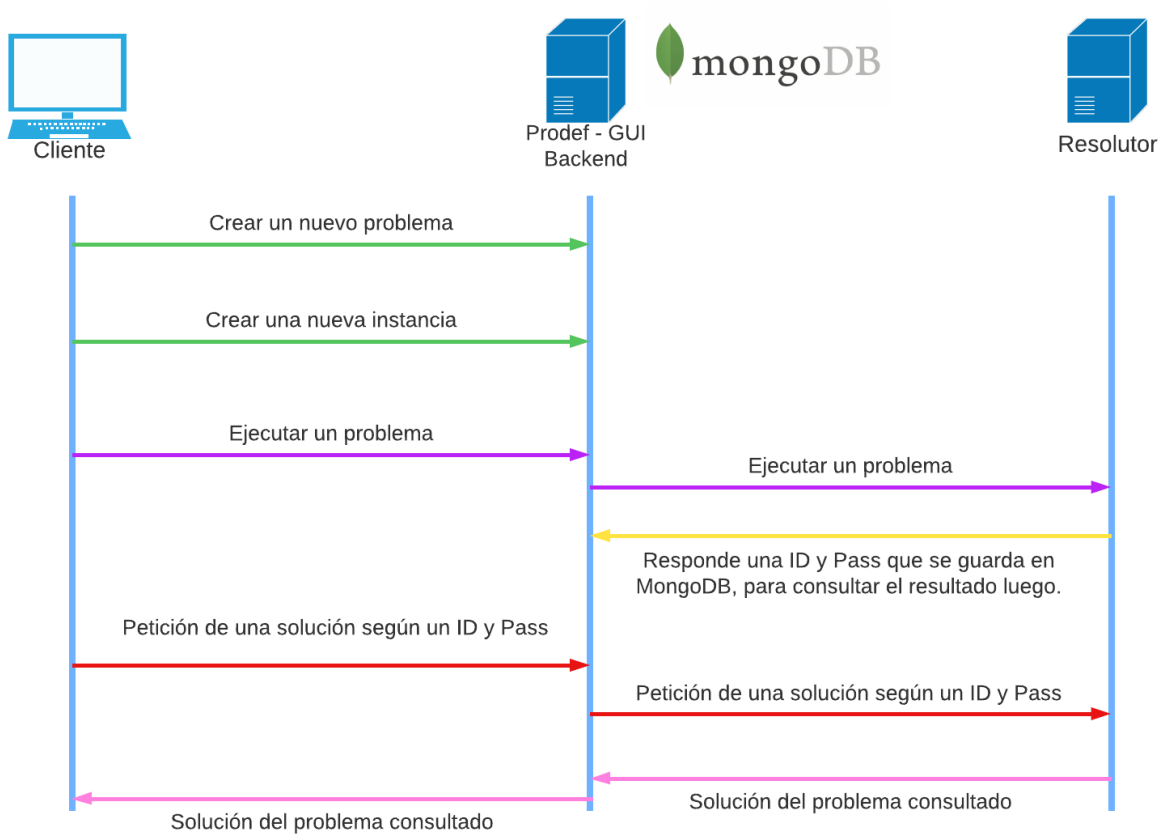


Figura 2.3: Esquema de la estructura interna Prodef

Capítulo 3

Problemas de Optimización

Para empezar con este capítulo vamos a necesitar introducir el término de 'Familia de Problemas', este se usa para poder agrupar a los distintos problemas que se analizarán a continuación. Esta agrupación se basa en diferentes atributos que posee la variable `result` dentro JSON resultante del problema. Los atributos que se tienen en cuenta para agrupar cada 'Familia de Problemas' se mencionarán más adelante cuando se vayan presentando.

3.1. Familia de Booleanos

Para llevar a cabo la agrupación de esta familia, tenemos en cuenta dos atributos del JSON, los cuales son, por un lado el `range.lowerBound` indicando el menor valor que puede tomar la variable, por otro lado el `range.upperBound`, que indica el mayor valor que puede alcanzar la variable. Pues si estos dos valores son 0 y 1 respectivamente, estamos ante un problema de booleanos.

Dentro de la familia de booleanos tenemos dos apartados, esto lo podemos saber por el atributo que posee el JSON `shape.type` que indica si es un vector o matriz.

3.1.1. Problema de la mochila

El problema de la mochila [13] comúnmente abreviado por KP, por su nombre en inglés (Knapsack problem); tiene como objetivo maximizar la cantidad de ítems que se puede meter dentro de un contenedor sin sobrepasar el peso máximo que permite dicho contenedor. Este problema está catalogado como un problema de vector booleano, debido a que esta dentro de está familia, y en el JSON el atributo `shape.type` es igual a *vector*.

Está es una posible solución [5].

Modelo matemático

Consideramos tener (n) cantidad de ítems, con un valor (v) y un peso (w) y un contenedor con una capacidad máxima de peso (W). Creamos una variable (x_i) que toma el valor de 1 cuando ese ítem está dentro del contenedor y 0 en otro caso.

La función objetivo consiste en maximizar la cantidad de ítems que se puede meter dentro de la mochila. En las restricciones tenemos que la cantidad de ítems dentro de la mochila no puede superar el peso máximo permitido y la variable x_i tiene que tomar un valor de 0 o 1.

El modelo matemático para este problema es el siguiente:

$$
\begin{aligned}
&\textit{max} \sum_{i=1}^{n} v_i x_i \\
&\textit{s.a.} \sum_{i=1}^{n} w_i x_i \leq W \\
&\quad x_i \in \{0,1\}, \quad i = 1, ..., n
\end{aligned}
$$

Definición en Prodef

A continuación veremos una parte que representa el resultado de este problema definido en Prodef.

```
{
  "symbol": "x",
  "value": [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
  "within": "integers",
  "name": "Items in the knapsack",
  "range": {
    "lowerBound": 0,
    "upperBound": 1
  },
  "shape": {
    "type": "vector",
    "isPermutation": false,
    "size": {
      "fixed": false,
      "value": "N"
    }
  }
}
```

Este fragmento es la variable que contiene realmente la solución al problema. En el cual se puede apreciar lo que se comentó al inicio, que para poder separar los problemas se utilizaban ciertos atributos del JSON (`range.lowerBound`, `range.upperBound` y `shape.type`).

No solo podemos ver eso, sino también el valor que toma la variable (*value*), donde cada posición representa un ítem, los cuales se encuentran almacenados en el atributo `object` del JSON completo, cuando el valor de una posición es 1 implica que ese elemento está seleccionado, y cuando es 0 es que este no pertenece a la solución óptima.

Bocetos

Una vez comprendido el resultado del problema lo siguiente que se hizo fue llevar a cabo un análisis para ver cuál sería la forma adecuada para representar este valor.

Primero se realizan un boceto de las posibles representaciones de la solución.

Como se aprecia en el primer boceto, imagen (Figura 3.1), se planteó la idea de separarlos en dos conjuntos donde uno tendría un borde verde, el cual representaba los ítems que si fueron seleccionados y otro con el borde rojo que tenía los ítems que no se seleccionaron.

Por otro lado se desarrolló otra idea, imagen (Figura 3.2), en la que se tenían todos los ítems juntos en una sola caja, y la forma de distinguir que ítems se seleccionaba y cuál no, era usando un color, que en un inicio se planteó un color verde claro para aquellos ítems que si estaban seleccionados y otro de color gris claro para los que no fueron seleccionados.

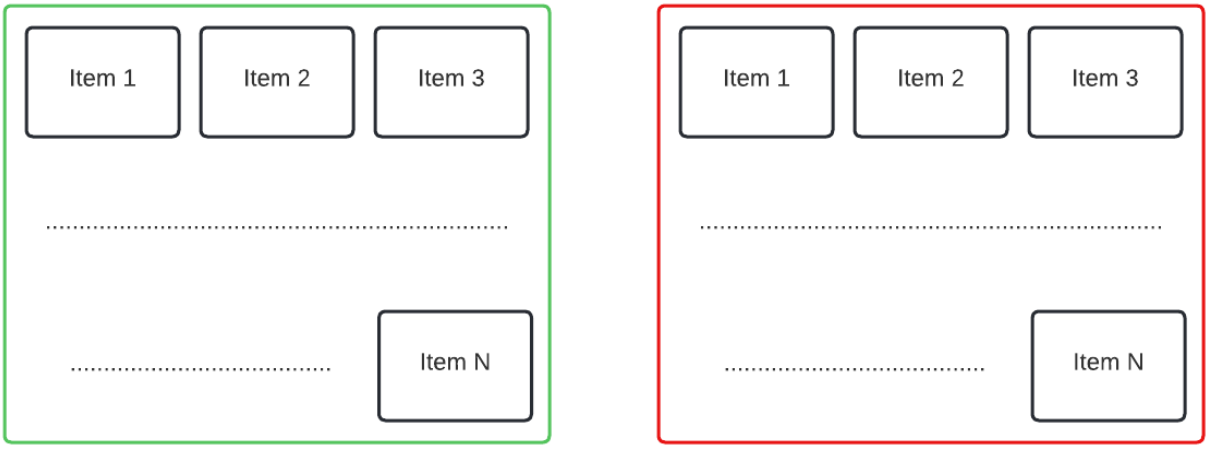


Figura 3.1: Primer boceto de la representación de un vector booleano

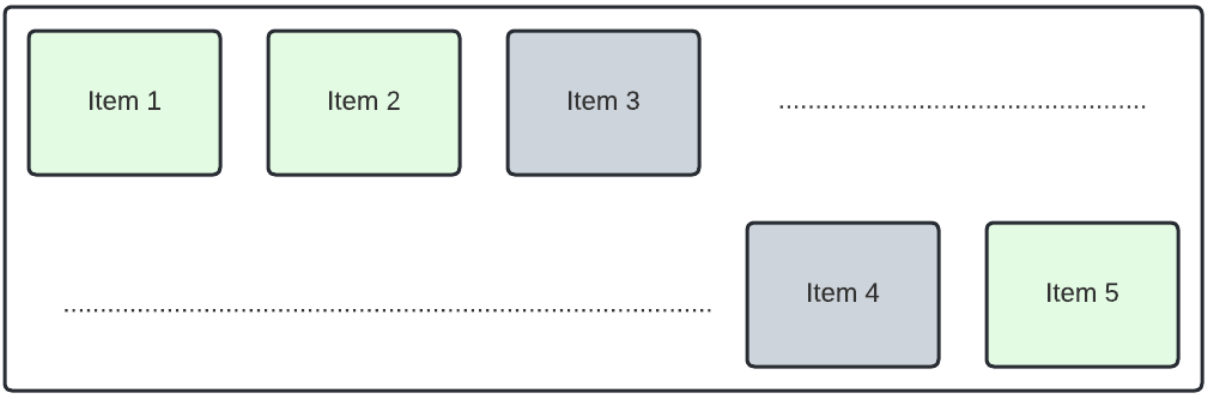


Figura 3.2: Segundo boceto de la representación de un vector booleano

3.1.2. Problema de Asignación 2D

El problema de asignación bi-dimensional [14] es un derivado del problema de transporte. El cual consiste en poder asignar una objeto a otro, en este caso trabajaremos con el ejemplo de profesores y asignaturas.

Está es una posible solución [6].

Modelo matemático

Partimos de que hay una cantidad de profesores (n), y un conjunto de asignaturas (m) donde se sabe el coste de cada asignación es (c_{ij}), definimos la variable x_{ij} como booleana, donde 1 significa que el profesor i, imparte la asignatura j, siendo 0 en otro caso.

La función objetivo consiste en reducir el coste de la asignación, la cuál está sujeta a que una asignatura la puede impartir un único profesor y un profesor puede tener una única asignatura y el valor que toma la variable x_{ij} solo puede ser 0 o 1.

El modelo matemático para este problema es el siguiente:

$$min \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}$$

$$s.a. \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, ..., m$$

$$\sum_{j=1}^{m} x_{ij} = 1, \quad i = 1, ..., n$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, ..., n, \quad j = 1, ..., m$$

Definición en Prodef

A continuación veremos una parte que representa el resultado de este problema definido en Prodef.

```
{
  "symbol": "x",
  "value": [
    [1,0,0,0,0],
    [0,1,0,0,0],
    [0,0,0,1,0],
    [0,0,1,0,0],
    [0,0,0,0,1]
  ],
  "within": "integers",
  "name": "Assigning a teacher to a subject",
  "range": {
    "lowerBound": 0,
    "upperBound": 1
  },
  "shape": {
    "type": "matrix",
    "isPermutation": false,
    "size": {
      "fixed": false,
```

```
      "value": "N"
    }
  }
}
```

Este fragmento es la variable que contiene realmente la solución al problema. En el cual se puede apreciar lo que se comentó al inicio, que para poder separar los problemas se utilizaban ciertos atributos del JSON (`range.lowerBound`, `range.upperBound` y `shape.type`).

No solo podemos ver eso, sino también el valor que toma dicha variable (`value`), donde apreciamos una matriz en la que en cada fila representa un profesor, teniendo en cuenta que estamos tratando con el problema mencionado en el modelo matemático, mientras que cada posición dentro de la fila, sería la asignatura que imparte este profesor. Las posiciones están relacionadas con las posiciones en las que se guardan en el atributo (`object`) del JSON completo.

Bocetos

Una vez comprendido el resultado del problema lo siguiente que se hizo fue llevar a cabo un análisis para ver cuál sería la forma correcta de poder representar este valor.

Primero se realizo un boceto de las posibles representaciones de la solución. Como se aprecia en el boceto, imagen (Figura 3.3), se planteó la idea de cada fila representarla como un contenedor con le nombre de lo que representase, y dentro de dicho contenedor incluir los ítems que están dentro de cada fila. A diferencia del problema anterior, que en el boceto se representaba todos los ítems dentro, en este se optó por solo representar los ítems que fueron solución óptima al problema.

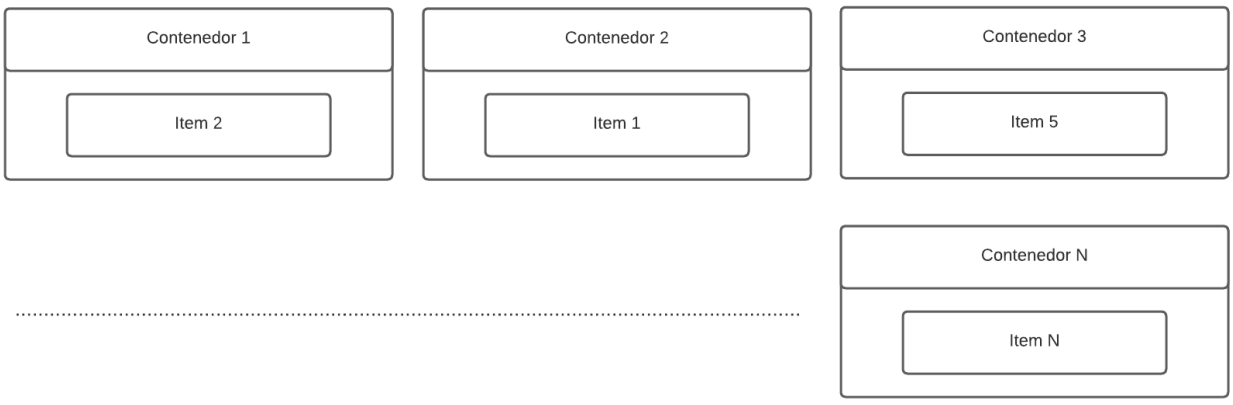


Figura 3.3: Boceto de la representación de una matriz booleana

3.2. Familia con Permutación

La forma de agrupar esta familia se basa en el atributo `shape.isPermutation`, el cual es booleano, decidiendo en caso de *true*, que se trata de un problema con permutación y en caso de *false*, un problema sin permutación, pues aquí nos interesa solo cuando este atributo está en *true*.

Dentro de la familia con permutación tenemos dos apartados, esto lo podemos saber por el atributo que posee el JSON `shape.type` que indica si es un vector o matriz.

3.2.1. TSP

El problema del viajante de comercio [15], también conocido como TSP por su nombre en inglés (Travelling Salesman Problem), es un problema de optimización combinatoria, que responde a la pregunta, cuál es el camino más corto que pase por un conjunto de ciudades. Este problema está catalogado como un problema de permutación con vector.

Está es una posible solución [7].

Modelo matemático

Teniendo en cuanta que hay una *n* cantidad de ciudades, con un coste para ir entre cada una de ellas c_{ij} y sea x_{ij} una variable que tendrá 1 si se toma el camino que va de *i* ciudad a *j* y 0 en otro caso.

La función objetivo de este problema consiste en minimizar el coste de recorrer todas las ciudades existentes, el cuál esta sujeto a que no se puede pasar de nuevo por una ciudad ya visitada y el valor de x_{ij} sea *0* o *1*.

El modelo matemático para este problema es el siguiente:

$$
\begin{aligned}
&\textit{min} \sum_{i=0}^{n} \sum_{i \neq j, j=0}^{n} c_{ij} x_{ij} \\
&\textit{s.a.} \sum_{i=0, i \neq j}^{n} x_{ij} = 1, \quad j = 0, ..., n \\
&\sum_{j=0, j \neq i}^{n} x_{ij} = 1, \quad i = 0, ..., n \\
&x_{ij} \in \{0, 1\}, \quad i, j = 0, ..., n
\end{aligned}
$$

Definición en Prodef

A continuación veremos una parte que representa el resultado de este problema definido en Prodef.

```
{
    "symbol": "city",
    "value": [0, 2, 4, 1, 3],
    "within": "integers",
    "name": "city",
    "range": {
      "lowerBound": "-Infinity",
      "upperBound": "Infinity"
    },
    "shape": {
      "type": "vector",
      "isPermutation": true,
      "size": {
        "fixed": false,
        "value": "N"
      }
```

```
17      }
18  }
```

Este fragmento es la variable que contiene realmente la solución al problema. En el cual se puede apreciar lo que se comentó al inicio, que para poder separar los problemas se utilizaban ciertos atributos del JSON (`shape.isPermutation` y `shape.type`).

No solo podemos ver eso, sino también el valor que toma dicha variable `value`, donde apreciamos un vector que contiene el conjunto de ciudades ordenadas que cumplen las condiciones de este problema. El valor que tiene cada posición del vector indica el objeto que está almacenado en el atributo `object` del JSON completo.

Bocetos

Una vez comprendido el resultado del problema lo siguiente que se hizo fue llevar a cabo un análisis para ver cuál sería la forma correcta de poder representar este valor.

Primero se realizó un boceto de las posibles representaciones de la solución. Como se aprecia en el boceto, imagen (Figura 3.4), se planteó la idea de hacer algo similar a un grafo, que representase a cada ciudad seleccionada como solución e indicar con una flecha cuál sería el camino a seguir.

Figura 3.4: Boceto de la representación de un vector con permutación

3.2.2. VRP

El problema de enrutamiento de vehículos [16], también conocido por VRP, por su nombre en inglés (Vehicle Routing Problem) es un problema de optimización combinatoria, el mismo es una variante del problema del viajante de comercio (TSP). Este problema resuelve la pregunta, ¿Cuál es el conjunto caminos óptimos para un grupo de vehículos que debe cumplir con las demandas de un grupo de clientes?. Este problema está catalogado como un problema de permutación con matriz.

Está es una posible solución [8].

Modelo matemático

Partiendo de un conjunto de ciudades n y un costo de viaje c_{ij} y una cantidad de vehículos disponibles m, se crea una variable x_{ij}, que es 1 si se toma la ruta de la ciudad i a la j, siendo 0 en otro caso.

La función objetivo consiste en minimizar el coste de las rutas, esto esta sujeto a ciertas restricciones, para empezar las 2 primeras quieren decir que solo pueden salir y entrar al punto de partida una m cantidad de vehículos; mientras que las otras dos, son iguales a las del TSP, haciendo que solo se pueda pasar por una ciudad una única vez.

El modelo matemático para este problema es el siguiente:

$$min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{\mathrm{ij}}x_{\mathrm{ij}}$$

$$s.a. \sum_{i=1}^{n} x_{\mathrm{i0}} = m$$

$$\sum_{j=1}^{n} x_{\mathrm{0j}} = m$$

$$\sum_{i=1}^{n} x_{\mathrm{ij}} = 1, \quad j = 0, ..., n$$

$$\sum_{j=1}^{n} x_{\mathrm{ij}} = 1, \quad i = 0, ..., n$$

$$x_{\mathrm{ij}} \in \{0, 1\}, \quad i, j = 0, ..., n$$

Definición en Prodef

A continuación veremos una parte que representa el resultado de este problema definido en Prodef.

```
{
    "symbol": "x",
    "value": [
      [0, 8, 6, 2, 5, 0],
      [0, 7, 1, 3, 0],
      [0, 9, 4, 0]
    ],
    "within": "integers",
    "name": "routes",
    "range": {
```

```
      "lowerBound": "-Infinity",
      "upperBound": "Infinity"
    },
    "shape": {
      "type": "matrix",
      "isPermutation": true,
      "size": {
        "fixed": false,
        "value": "N"
      }
    }
}
```

Este fragmento es la variable que contiene realmente la solución al problema. En el cual se puede apreciar lo que se comentó al inicio, que para poder separar los problemas se utilizaban ciertos atributos del JSON (`shape.isPermutation` y `shape.type`).

No solo podemos ver eso, sino también el valor que toma dicha variable `value`, donde se aprecia que es una matriz, en la que cada fila, representaría un vehículo y cada posición dentro de dicha fila, indican las posiciones de las ciudades que visita este vehículo en orden. Las posiciones están relacionadas con la forma en la que se guardan los datos en el atributo `object` del JSON completo.

Bocetos

Una vez entendido lo que representan los datos de la solución, lo suyo sería empezar a pensar como queríamos representar esa información y es así como pasamos al desarrollo de unos bocetos.

Como se aprecia en la imagen (Figura 3.5), se extrapoló la representación del problema de Asignación 2D, donde se crearon contenedores que representan un vehículo, y dentro del mismo, seguimos el diseño empleado para el TSP, que como se mencionó, era representar a cada ciudad seleccionada como solución e indicar con una flecha cuál sería el camino a seguir.

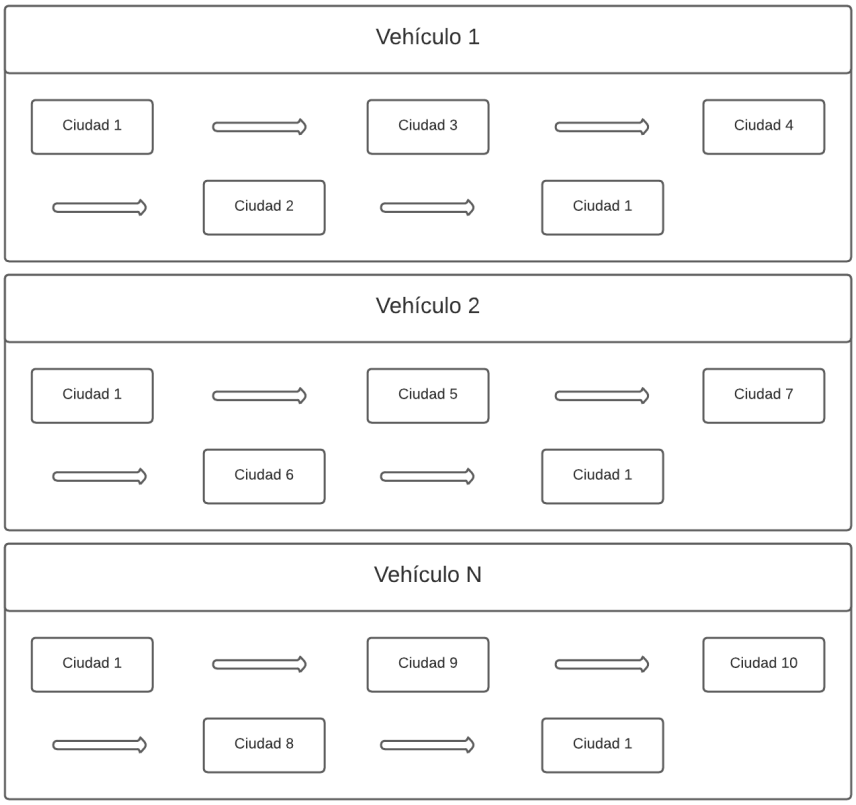


Figura 3.5: Boceto de la representación de una matriz con permutación

3.3. Familia Enteros

La forma de agrupar esta familia se basa en los atributos `shape.isPermutation`, `range.lowerBound` y `range.upperBound`, que son los mismos que tenemos en cuenta para las otras dos familias, solo que en esta familia tomamos los casos contrarios, o sea buscamos que no se a de permutación y que su límite inferior sea distinto de 0 y su límite superior distinto de 1.

Dentro de esta familia tenemos la representación de matrices.

3.3.1. Problema de Asignación 3D

Un problema de asignación 3D [16] consiste en intentar optimizar los recursos asignándolos a de tal forma que minimice el costo de dicha asignación, suele usarse mucho para repartir tareas. Un ejemplo de esto puede ser el de asignar asignaturas a profesores en unas determinadas aulas.

Está es una posible solución [9].

Modelo matemático

Suponiendo que tenemos un conjunto de profesores m, otro conjunto de asignaturas w y un grupo de aulas t; sabemos el coste que tiene asignar una asignatura a un profesor en un aula es definido como c_{ijk}; ahora definimos una variable x_{ijk} que toma valor 1 si la asignatura i se asigna a el profesor j en el aula k.

La función objetivo consiste en minimizar el coste de asignación, la misma esta sujeta

a ciertas restricciones, como son, que cada profesor en cada aula puede tener una única asignatura, otra sería que cada profesor con cada asignatura puede tener una única aula y la última que cada aula con cada asignatura puede ser impartida por un profesor.

El modelo matemático para este problema es el siguiente:

$$
\begin{aligned}
&\textit{min} \sum_{i=0}^{m}\sum_{j=0}^{t}\sum_{k=0}^{w} c_{\text{ijk}}x_{\text{ijk}} \\
&\textit{s.a.} \sum_{i=0}^{m}\sum_{j=0}^{t} x_{\text{ijk}} = 1, \quad k = 0, ..., w \\
&\quad \sum_{i=0}^{m}\sum_{j=0}^{w} x_{\text{ijk}} = 1, \quad k = 0, ..., t \\
&\quad \sum_{i=0}^{w}\sum_{j=0}^{t} x_{\text{ijk}} = 1, \quad k = 0, ..., m \\
&\quad x_{\text{ijk}} \in \{0, 1\}, \quad i = 0, ..., m, \quad j = 0, ..., t, \quad k = 0, ..., w
\end{aligned}
$$

Definición en Prodef

A continuación veremos una parte que representa el resultado de este problema definido en Prodef.

```
{
  "symbol": "x",
  "value": [
    [-1,1,-1],
    [0,-1,-1],
    [-1,-1,2]
  ],
  "within": "integers",
  "name": "Assignment teacher, classroom, subject",
  "range": {
    "lowerBound": "-Infinity",
    "upperBound": "Infinity"
  },
  "shape": {
    "type": "matrix",
    "isPermutation": false,
    "size": {
      "fixed": false,
      "value": "N"
    }
  }
}
```

Este fragmento es la variable que contiene realmente la solución al problema. En el mismo se ve lo que se comentó al inicio, que para poder separar los problemas se utilizaban ciertos atributos del JSON (`shape.isPermutation`, `range.lowerBound` y `range.upperBound`).

No solo podemos ver eso, sino también el valor que toma dicha variable `value`, donde tenemos una matriz en la que el índice de cada fila indica el profesor, el índice de la

Profesor 1		Asignatura 3		
Profesor 2			Asignatura 2	
Profesor 3	Asignatura 1			
Profesor 4				Asignatura 4
	Aula 1	Aula 2	Aula 3	Aula 4

Figura 3.6: Boceto de la representación de una matriz de enteros

columna el aula y el valor que toma cada celda, es la posición de la asignatura que se asigna, en caso de que la celda tenga un valor negativo (*-1*), esto significa que en esta no se ha asignado ninguna asignatura. Las posiciones están relacionadas con la forma en la que se guardan los datos en el atributo `object` del JSON completo.

Bocetos

Una vez entendido lo que representan los datos de la solución, lo suyo sería empezar a pensar como queríamos representar esa información y es así como pasamos al desarrollo de unos bocetos.

Como se aprecia en la imagen (Figura 3.6) se planteó la idea de representarlo como la propia matriz, esto surgió como idea base de la imagen (Figura 1.3).

Capítulo 4

Prodef Solution

Como se viene comentando con anterioridad, la necesidad de una interfaz gráfica para la visualización de resultados era algo inminente, pues en este capítulo, explicaré el trabajo realizado.

4.1. Tecnologías

Al querer realizar una interfaz web para la visualización de datos, en lo primero a pensar es con que herramientas vas a querer desarrollar el proyecto, en este caso se optó por usar la librería de React+17 [10], que junto la herramienta de gestión de estado, Redux [11], se pueden crear grandes productos.

En lo segundo que hay que pensar es en que lenguaje lo vas a crear, pues en este proyecto, se optó por desarrollarlo en JavaScript, esto debido a que es un lenguaje capaz de ejecutarse en navegadores y muy simple de manejar.

Para el desarrollo se usaron varias librerías, una de las más importantes es Material UI (MUI) [12], ya que fue la que me facilitó el diseño de la página y todos sus componentes, debido a que es una biblioteca de componentes especifica para usar con React.

4.2. Uso

Para comenzar a usar la interfaz desarrollada, es necesario tener un JSON con el resultado de un problema, cuyo JSON tiene que seguir la estructura de ProdefLang, para el correcto funcionamiento de la interfaz. Ejemplo como el del Apéndice A.1 y/o como algunos de los ejemplos que se mostraron en el capítulo anterior.

Una vez teniendo esto en cuenta, empezaremos a ver como funciona toda la interfaz.

Lo primero es subir este archivo JSON del que hablamos, para ello en la interfaz hay un botón en la parte superior derecha, el cual se aprecia en la Figura 4.1.

Figura 4.1: Página inicial de la interfaz

Al pulsar en él, se nos abre una ventana (Figura 4.2) para poder seleccionar el archivo JSON que queremos subir. Una vez seleccionado el archivo, este se añade a la interfaz (Figura 4.3) para así poder trabajar con el mismo.

La interfaz permite subir más de un resultado para visualizarlo (Figura 4.4).

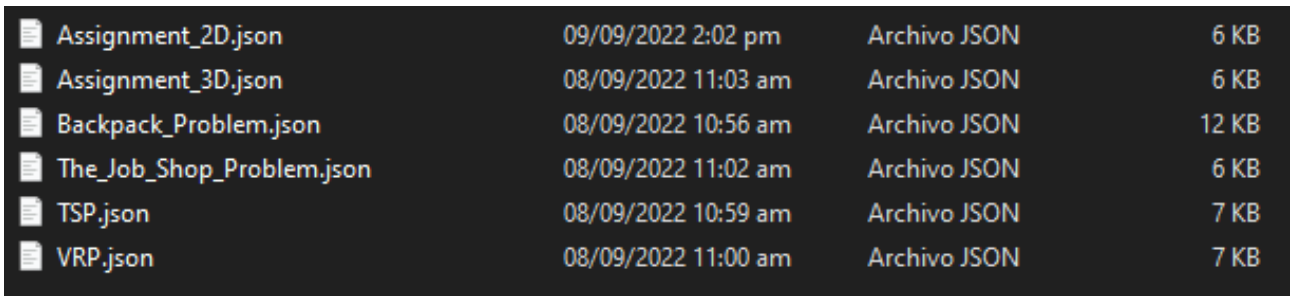


Figura 4.2: Listado de problemas

Figura 4.3: Problema cargado

Figura 4.4: Múltiples problemas cargados

Para decidir que representación usar en cada uno de los problema de ejemplos, podemos apoyarnos en este esquema (Figura 4.5) que posee algunos ejemplos de problemas.

Una vez cargado el problema o el conjunto de problemas, para poder visualizarlo, basta con hacer clic sobre su nombre, para que así aparezca más información del mismo (Figura 4.6).

Ahora iremos describiendo todo lo que tiene esta representación. Para empezar tenemos en la parte superior, debajo de la barra de navegación, encontramos el nombre del problema que estamos visualizando y un poco más a la derecha un selector de resultados, el cual permite cambiar entre los distintos resultados que se han obtenido de este problema.

Seguido tenemos una serie de datos acerca del estado del problema, en el cual podemos apreciar el tiempo que tardo en resolver el problema, el estado de ejecución del problema, para saber si ya está resuelto o este pendiente, el resultado seleccionado, también muestra si el problema es fiable, y la última actualización del problema.

Después de la fina linea, un poco más abajo encontramos la representación gráfica del resultado del problema tomado como ejemplo para poder explicar el uso de la herramienta.

Para empezar nos encontramos con unos selectores, uno de ellos nos permite seleccionar el atributo de los objetos que tiene definido el problema, en este problema en concreto tenemos el atributo tiempo, que indica el tiempo necesario que hay que impartir esta asignatura, y el atributo coste, el cual indica el precio a pagar por matricular esta asignatura (Figura 4.7), mientras que otro nos permite seleccionar como queremos que se modifique la representación (Figura 4.8). Para aplicar esta selección, tendremos que pulsar en el botón que dice 'Apply' que hará efectivo la selección, en caso de querer reiniciar el estilo que tiene aplicado, basta con pulsar el botón de 'Reset'. Más adelante se mostrará como se ven reflejadas estas acciones en la representación.

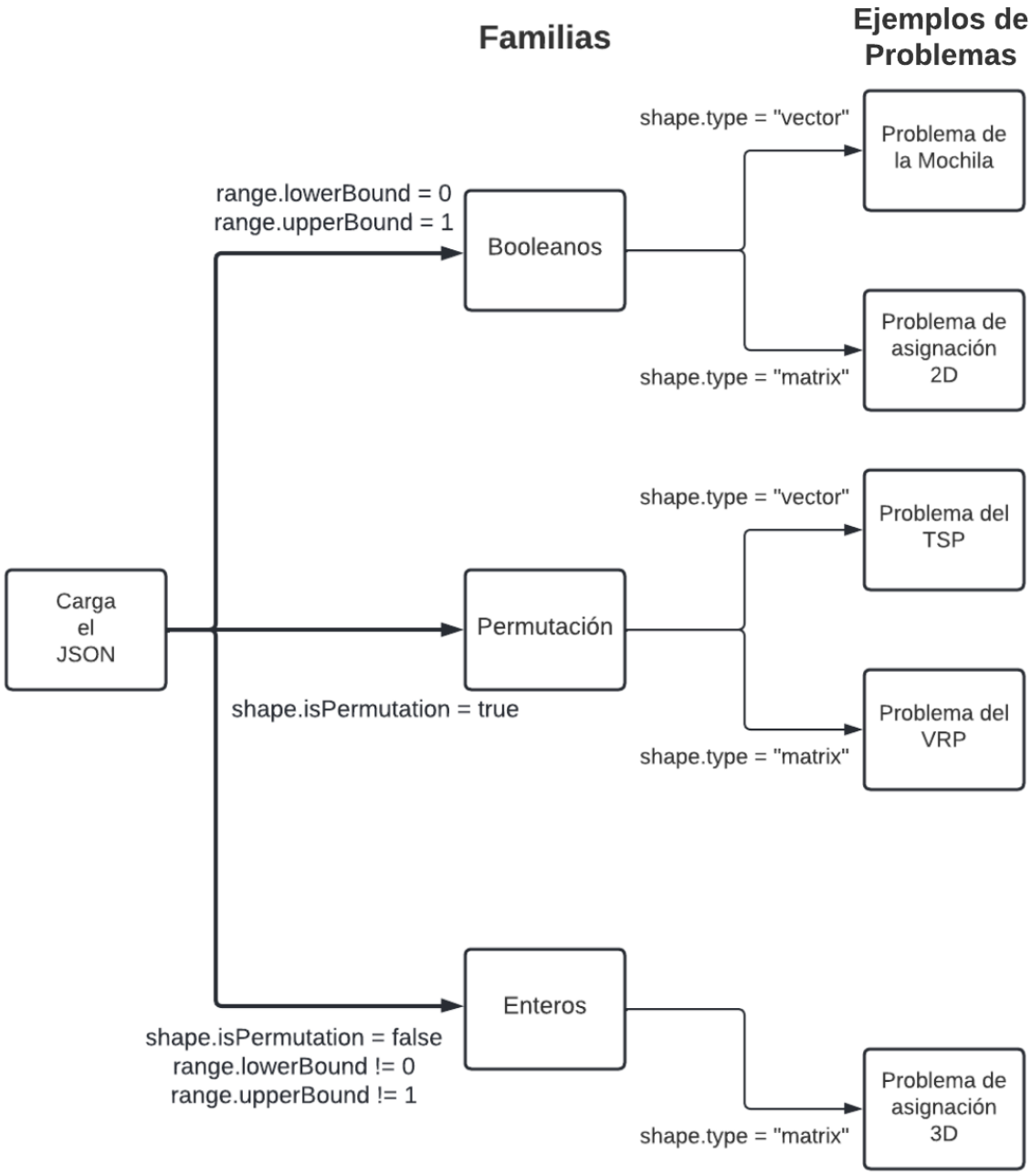


Figura 4.5: Diagrama de como se distinguen las familias de problemas

A continuación vamos a ver la representación del resultado del problema. Al inicio de esta representación tenemos el nombre de la variable, para así saber que estamos representando, seguido de esta viene la representación gráfica de la solución.

En esta representación se puede apreciar como al ’Profesor 1’ se le asignó la ’Asignatura 1’. Extrapolando esto a los otros profesores, se refleja el resultado del problema.

Esto es así para el resultado 1, pero este problema posee dos resultados, para poder ver el otro resultado (Figura 4.9), se explicó anteriormente como cambiar de resultado, solo tienes que cambiar el selector de la parte de arriba a la derecha, esto generara una actualización de los datos y se mostraran los datos del resultado seleccionado.

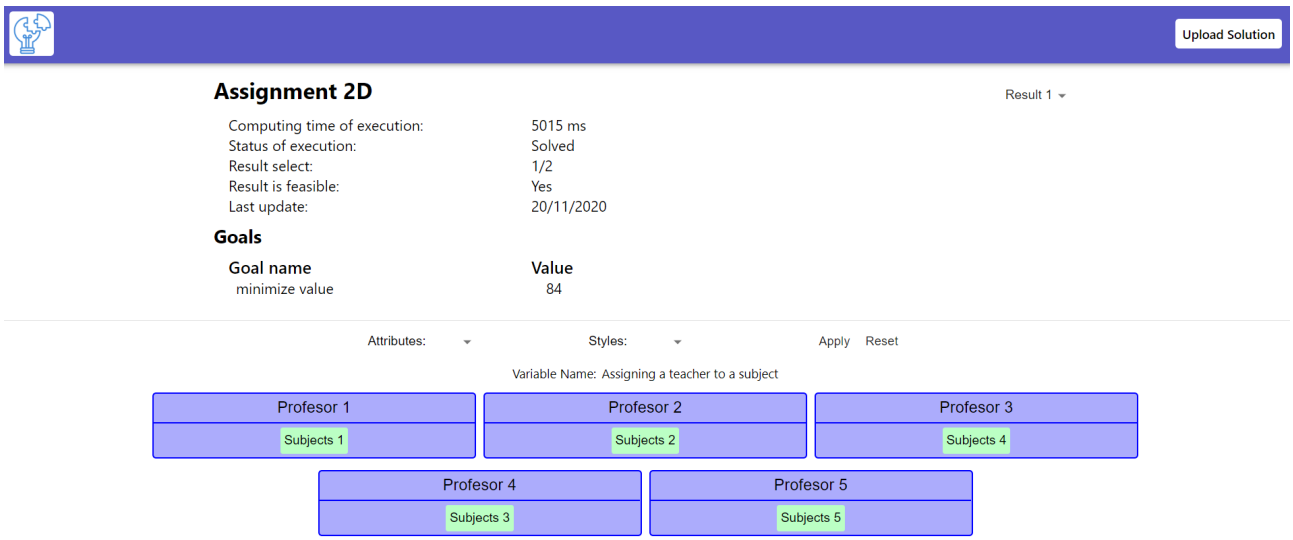


Figura 4.6: Representación de un problema

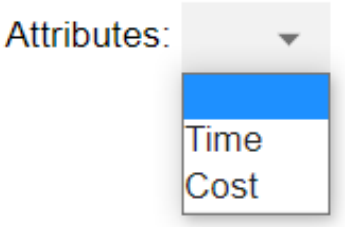


Figura 4.7: Selector de atributos

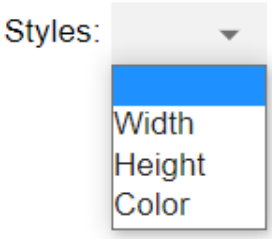


Figura 4.8: Selector de estilos

Pero esto no es todo, como se mencionó anteriormente, la interfaz permite modificar la visualización de los resultados basándose en los campos seleccionados en los selectores (Figura 4.7 y Figura 4.8). Ahora se mostraran ejemplos de como se ven los resultados con cada combinación de los selectores y se explicaran que significa cada una.

Tiempo y ancho

Esta combinación en los selectores lo que hará será tomar el valor del atributo tiempo, que ya se explicó lo que representaba, que tiene el objeto, en este caso asignatura, y basándonos en este valor, modificara el ancho de la visualización (Figura 4.10).

Tiempo y alto

Esta combinación en los selectores lo que hará será tomar el valor del atributo tiempo que tiene el objeto, en este caso asignatura, y basándose en este valor, modificara el alto

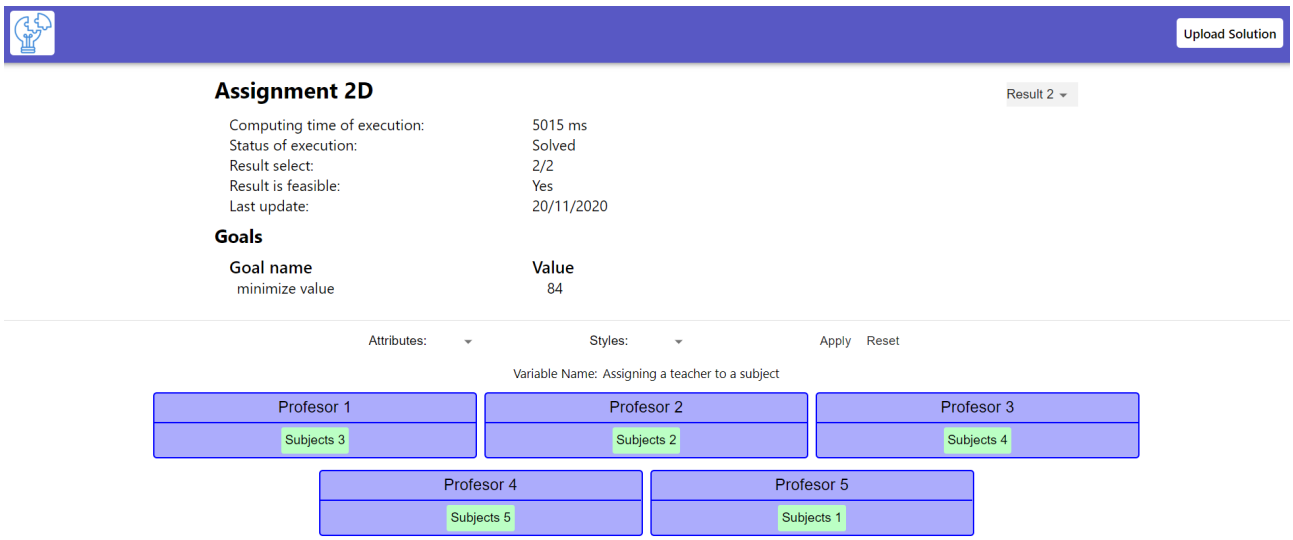


Figura 4.9: Representación gráfica de la solución 2

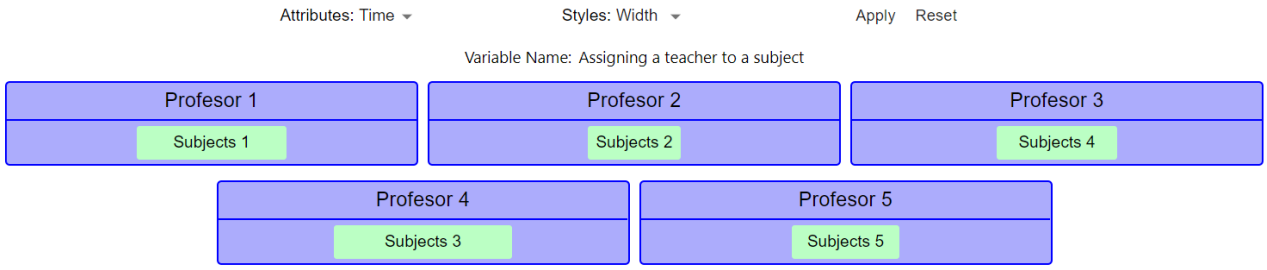


Figura 4.10: Problema de asignación 2D modificado(tiempo/ancho)

de la visualización (Figura 4.11).

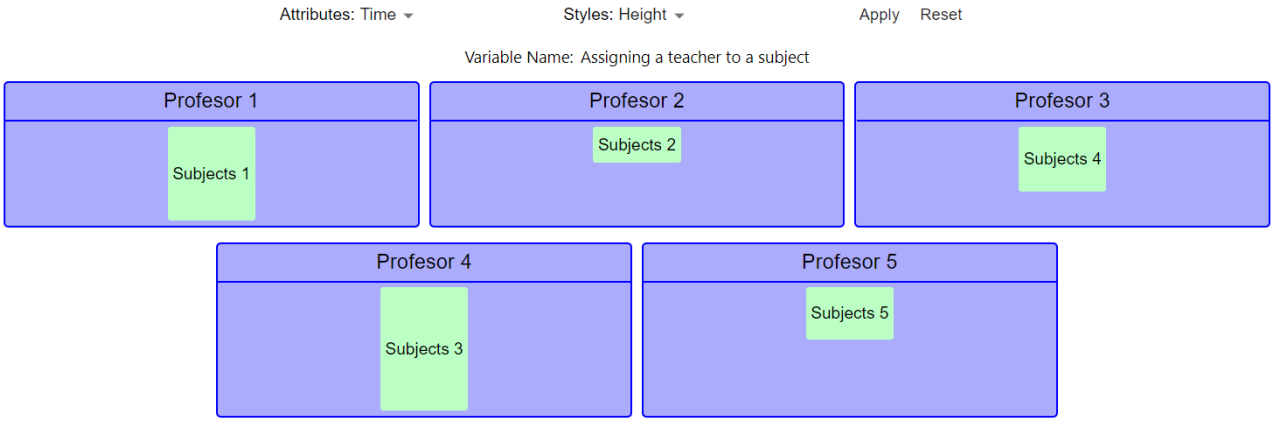


Figura 4.11: Problema de asignación 2D modificado(tiempo/alto)

Tiempo y color

Esta combinación en los selectores lo que hará será tomar el valor del atributo tiempo que tiene el objeto, en este caso asignatura, y en relación con este valor, modificara el color de la visualización (Figura 4.12), en una escala de rojo a verde (Figura 4.13), donde el rojo significaría 0 y el verde el 100.

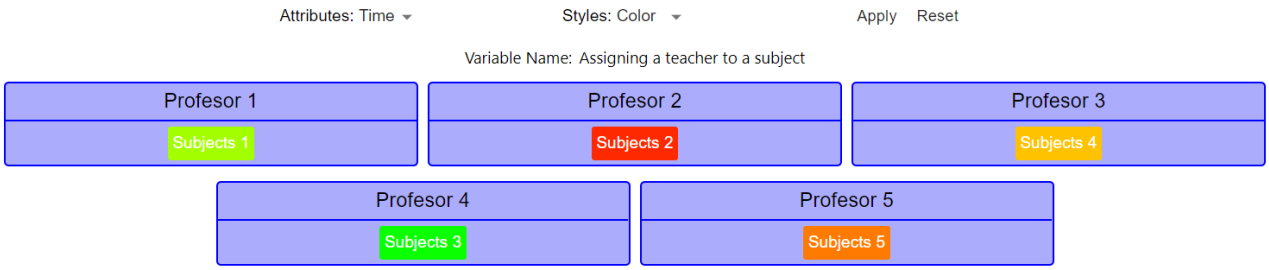


Figura 4.12: Problema de asignación 2D modificado(tiempo/color)

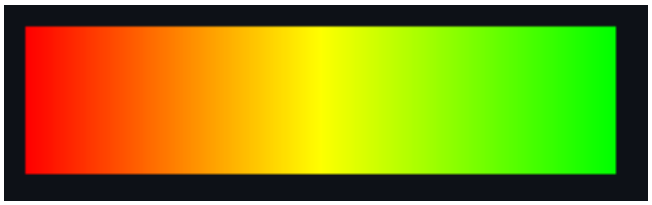


Figura 4.13: Escala de color

Coste y ancho

Esta combinación en los selectores lo que hará será tomar el valor del atributo coste que tiene el objeto, en este caso asignatura, y en base a este valor, modificara el ancho de la visualización (Figura 4.14).

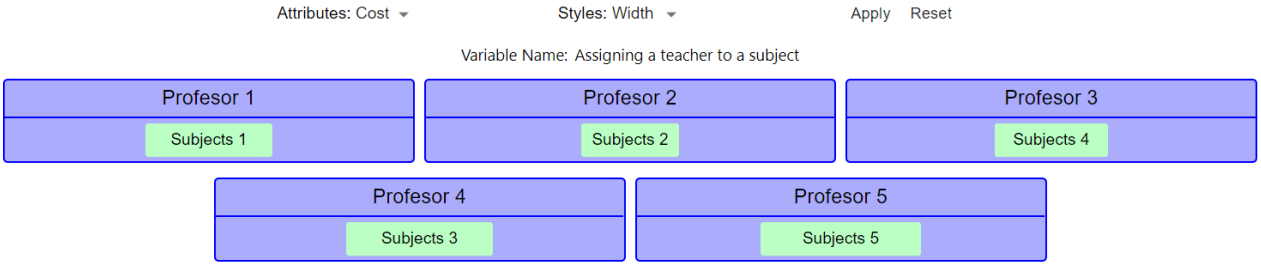


Figura 4.14: Problema de asignación 2D modificado(coste/ancho)

Coste y alto

Esta combinación en los selectores lo que hará será tomar el valor del atributo coste que tiene el objeto, en este caso asignatura, y basándonos en este valor, modificara el alto de la visualización (Figura 4.15).

Coste y color

Esta combinación en los selectores lo que hará será tomar el valor del atributo coste que tiene el objeto, en este caso asignatura, y en base a este valor, modificara el color de la visualización (Figura 4.16), en una escala de rojo a verde (Figura 4.13), donde el rojo significaría 0 y el verde el 100.

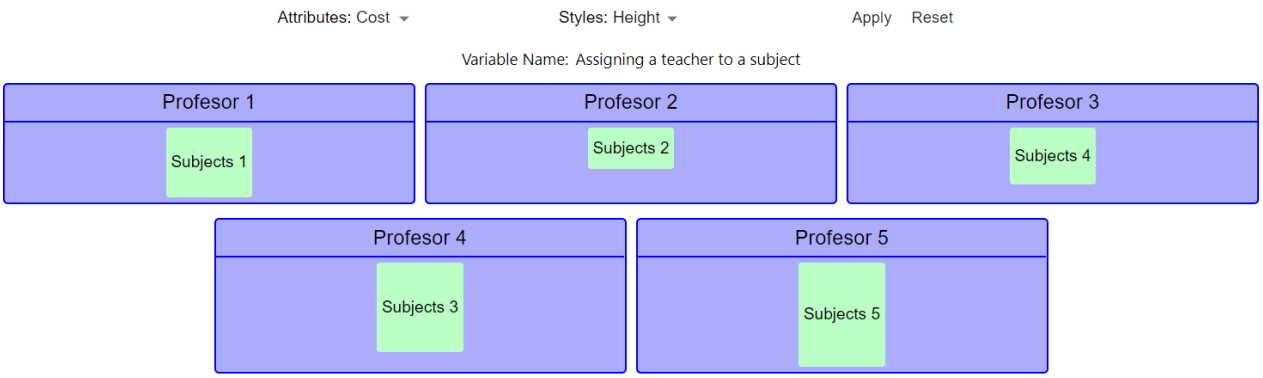


Figura 4.15: Problema de asignación 2D modificado(coste/alto)

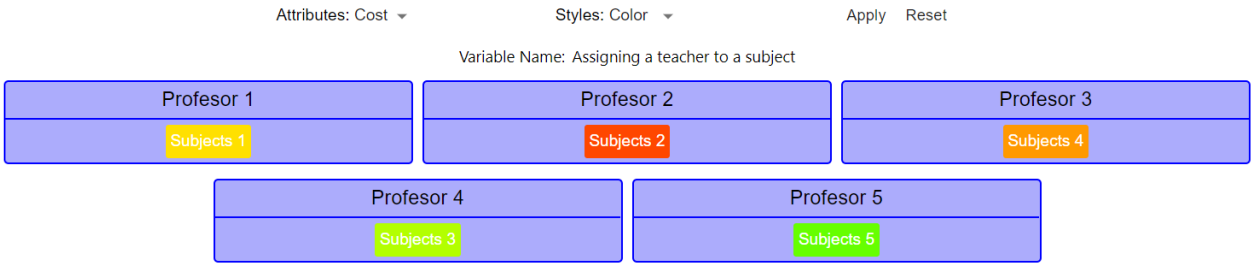


Figura 4.16: Problema de asignación 2D modificado(coste/color)

Mezcla de estilos

Como se ha visto anteriormente, se puede modificar la representación basándose en los atributos y estilos seleccionados, pero si se han dado cuenta se ha hablado de un atributo junto a un estilo, pues en este punto, se mostrara que se pueden tener una mezcla de estilos, por ejemplo como se aprecia en la Figura 4.17 hay una mezcla de estilos, donde están activos el par coste-color y tiempo-ancho, pues convenciones como estas están permitidas, para que el usuario pueda visualizar distintos atributos a la vez.

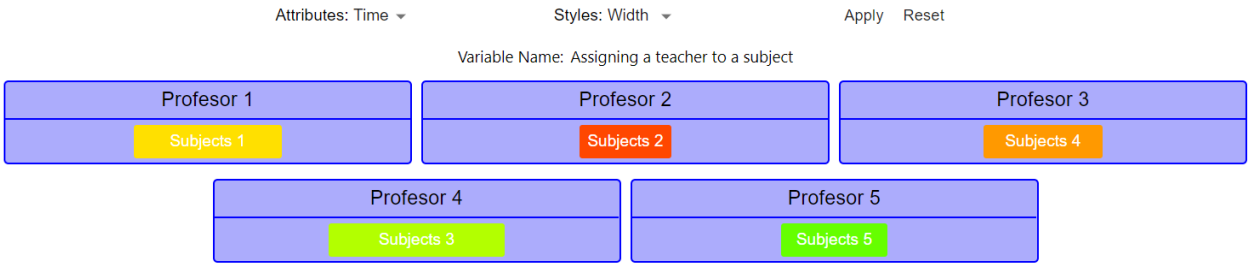


Figura 4.17: Problema de asignación 2D modificado(coste/color y tiempo/ancho)

Otro ejemplo podría ser coste-alto y tiempo-ancho, como se aprecia en la Figura 4.18

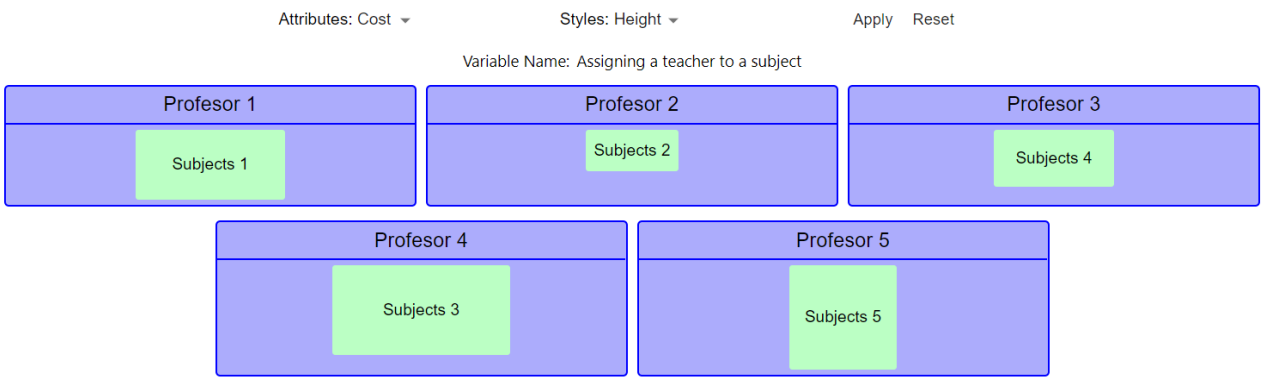


Figura 4.18: Problema de asignación 2D modificado(coste/alto y tiempo/ancho)

Capítulo 5

Resultados

En este capítulo mostraremos como se representan todos los problemas usados para el análisis. Estos son algunos ejemplos a soluciones de problemas.

5.1. Problema de la mochila

A continuación veremos la representación gráfica de la solución al problema de la mochila el cual se definió en el Capítulo 3.

La primera Figura 5.1 ilustra la representación del resultado 1 de este problema, donde se esta aplicando el estilo ancho con el atributo valor del ítem, mientras que la Figura 5.2 muestra el resultado 2, que está usando el estilo alto con el atributo peso.

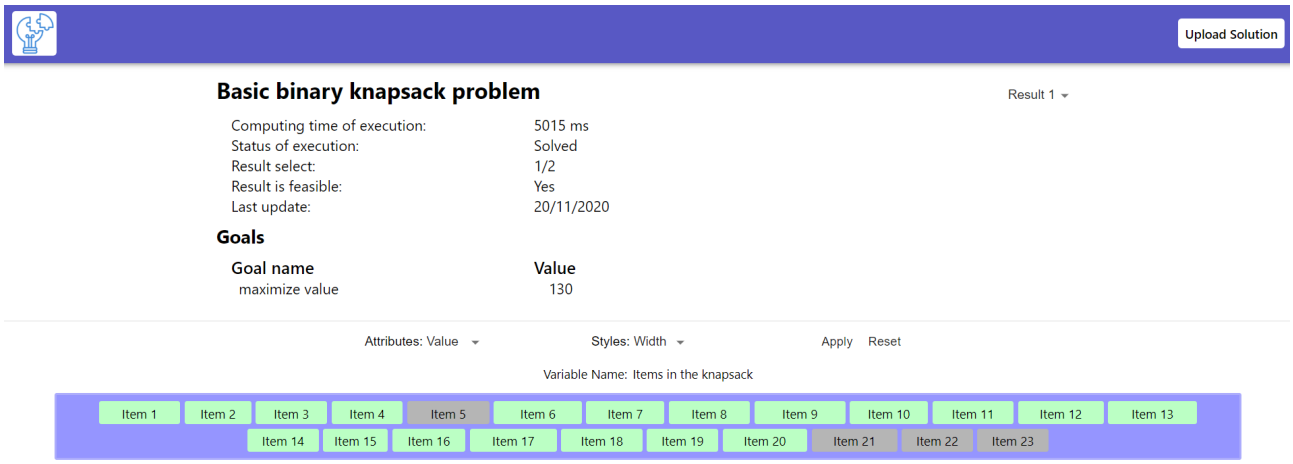


Figura 5.1: Problema de la mochila: resultado 1

5.2. Problema de asignación 2D

Ahora veremos la representación gráfica de la solución al problema de asignación 2D el cual se definió en el Capítulo 3.

La primera Figura 5.3 se aprecia la representación del resultado 1 de este problema, en el cual se esta aplicando el estilo color y el atributo coste, mientras que la Figura 5.4 se ve el resultado 2, en la cual se aplica el estilo alto y el atributo tiempo.

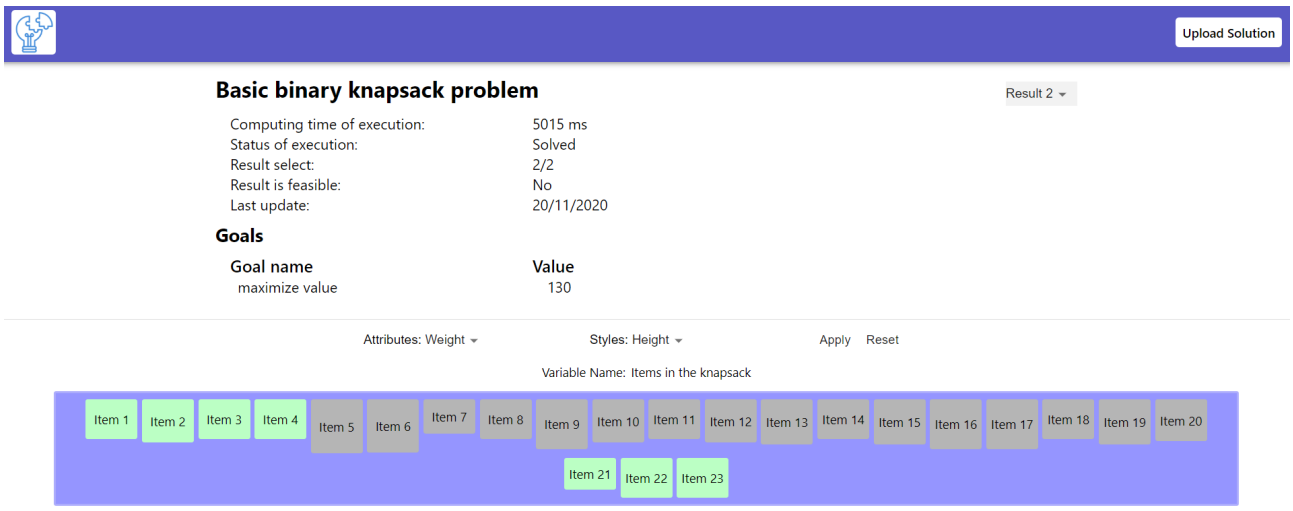


Figura 5.2: Problema de la mochila: resultado 2

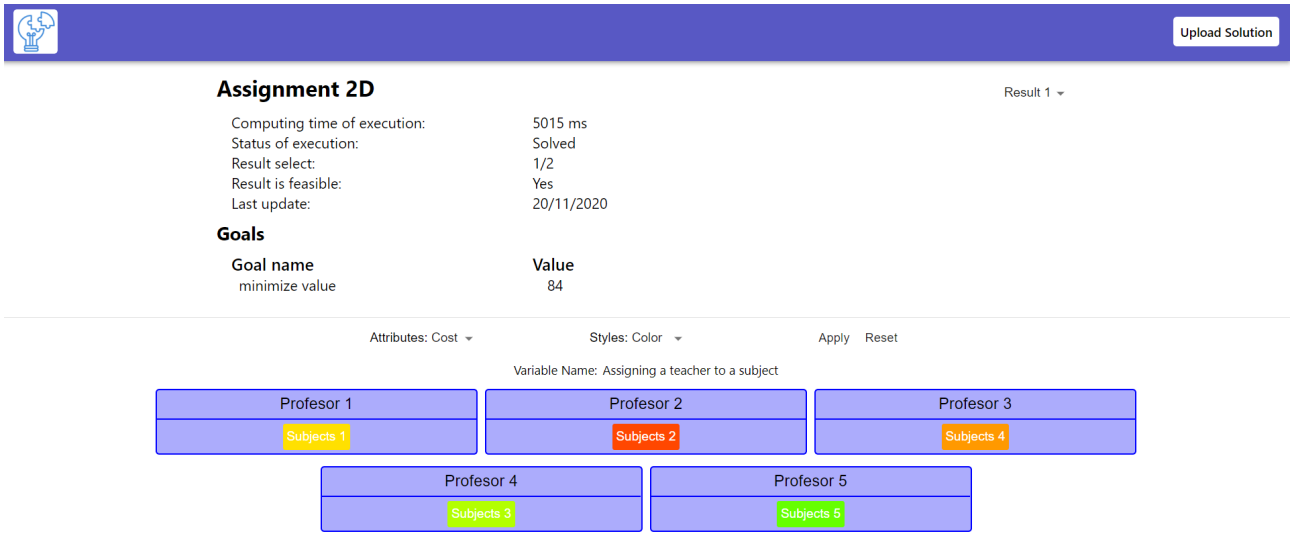


Figura 5.3: Problema de asignación 2D: resultado 1

5.3. Problema del TSP

A continuación veremos el resultado gráfico de la solucion al problema del TSP el cual se definió en el Capítulo 3.

La primera Figura 5.5 se ve la representación del resultado 1 de este problema, donde se está usando el estilo alto con el atributo largo de la ciudad, mientras que la Figura 5.6 ilustra el resultado 2, donde se usa el estilo color y el atributo largo.

En este problema en particular, se le añadió algo más a la representación que no se encontraba en los bocetos, por ende voy a explicar que es lo añadido, si miraron las figuras anteriores mencionadas, se pudieron dar cuenta que encima de cada flecha existe un número, este número no se encontraba en el diseñó inicial, el mismo indica la distancia que hay entre las ciudades.

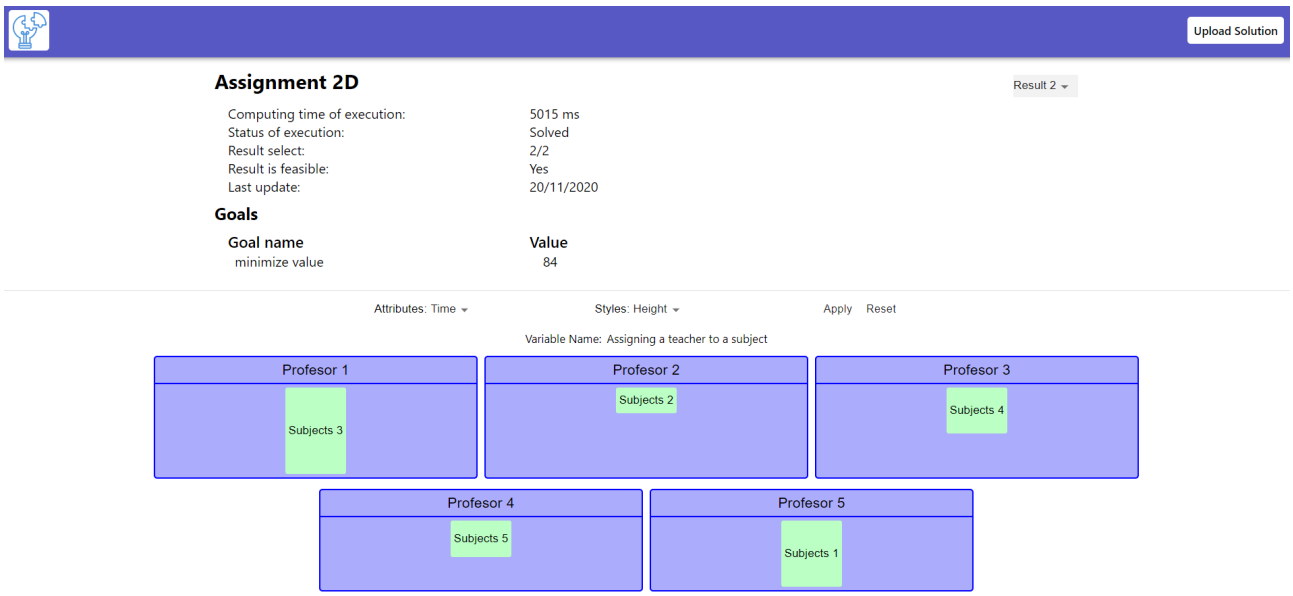


Figura 5.4: Problema de asignación 2D: resultado 2

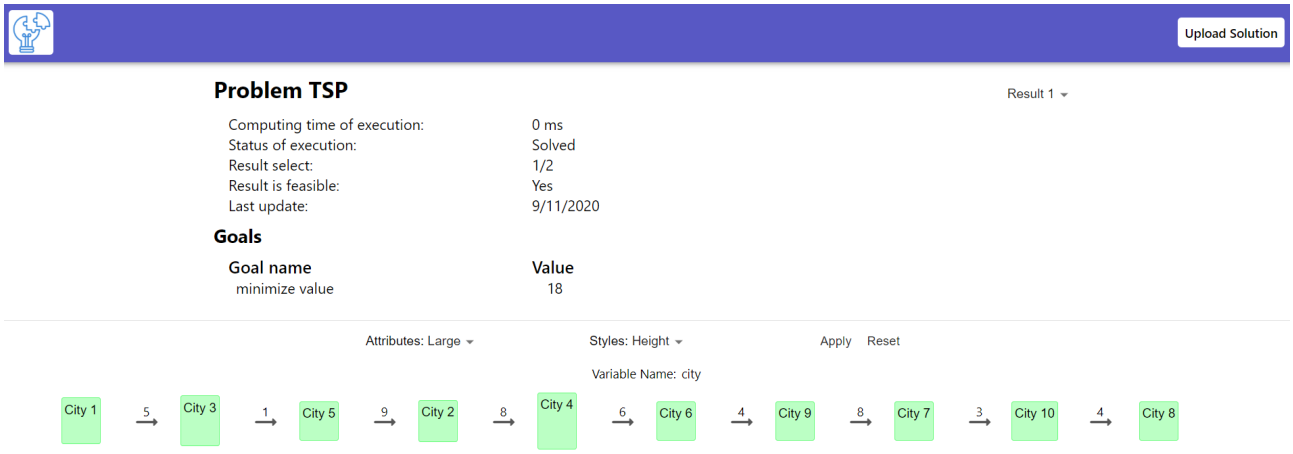


Figura 5.5: Problema del TSP: resultado 1

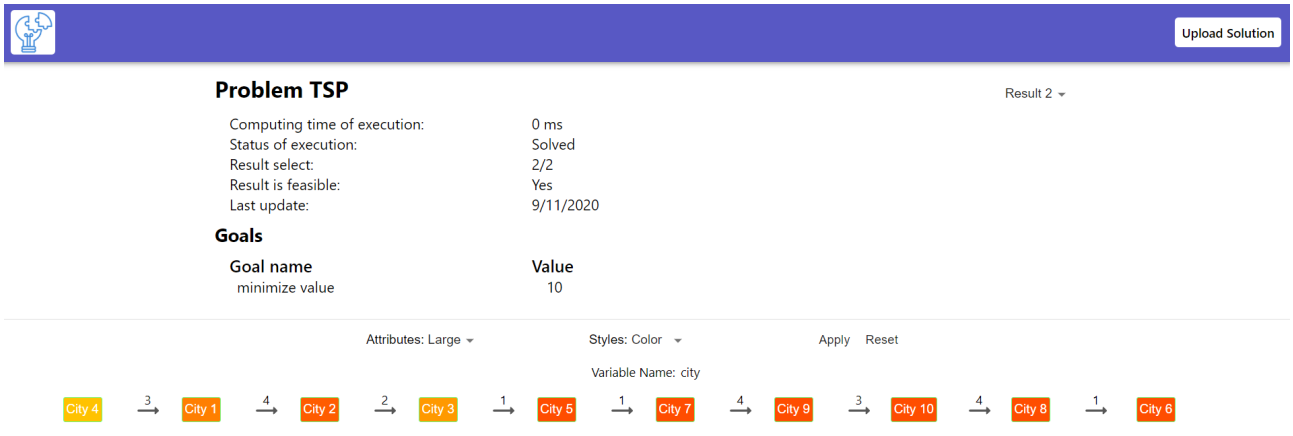


Figura 5.6: Problema del TSP: resultado 2

5.4. Problema del VRP

Ahora veremos la representación gráfica al problema del VRP el cual se definió en el Capítulo 3.

La primera Figura 5.7 se ve la representación del resultado 1 de este problema, haciendo uso del estilo alto y el atributo largo, mientras que la Figura 5.8 ilustra el resultado 2, se usa el atributo largo y el estilo color.

Al igual que con el TSP, en el boceto de este problema no se diseñó la funcionalidad de indicar las distancias entre ciudades, pero a lo largo del desarrollo se implementó.

Figura 5.7: Problema del VRP: resultado 1

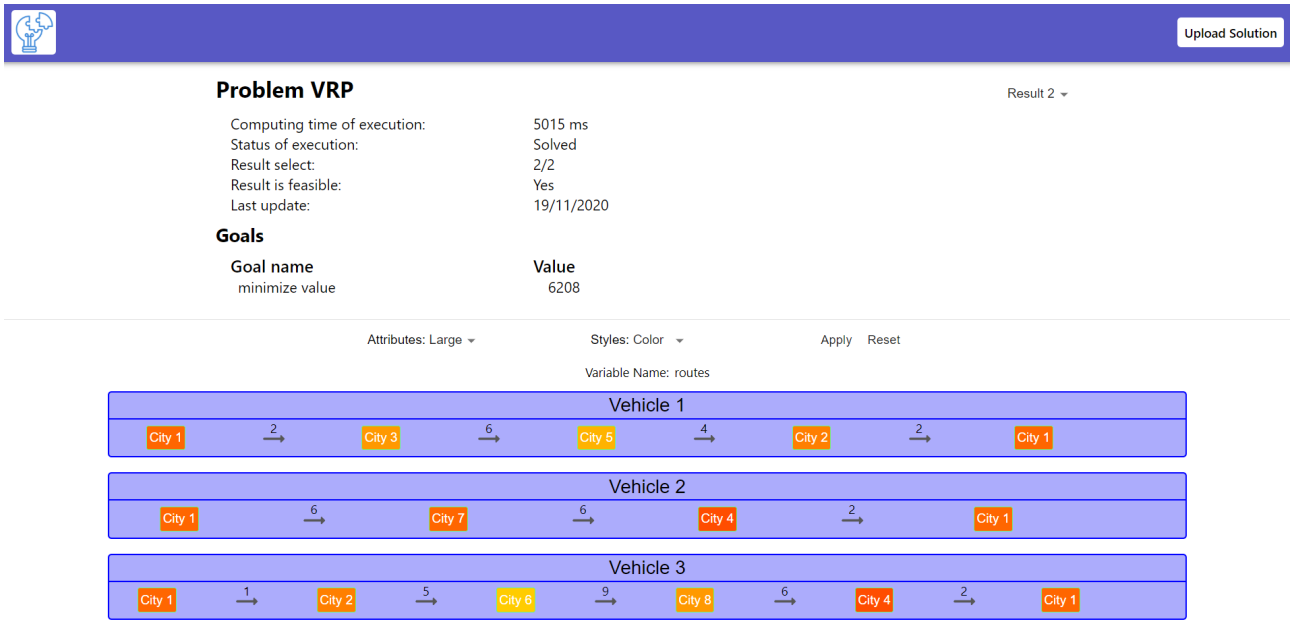


Figura 5.8: Problema del VRP: resultado 2

5.5. Problema de asignación 3D

A continuación veremos el resultado gráfico a la solución del problema de asignación 3D el cual se definió en el Capítulo 3.

La primera Figura 5.9 se ve la representación gráfica del resultado 1 de este problema, que tiene aplicado el estilo de ancho con el atributo información, mientras que la Figura 5.10 se aprecia el resultado 2, con estilo color y atributo información.

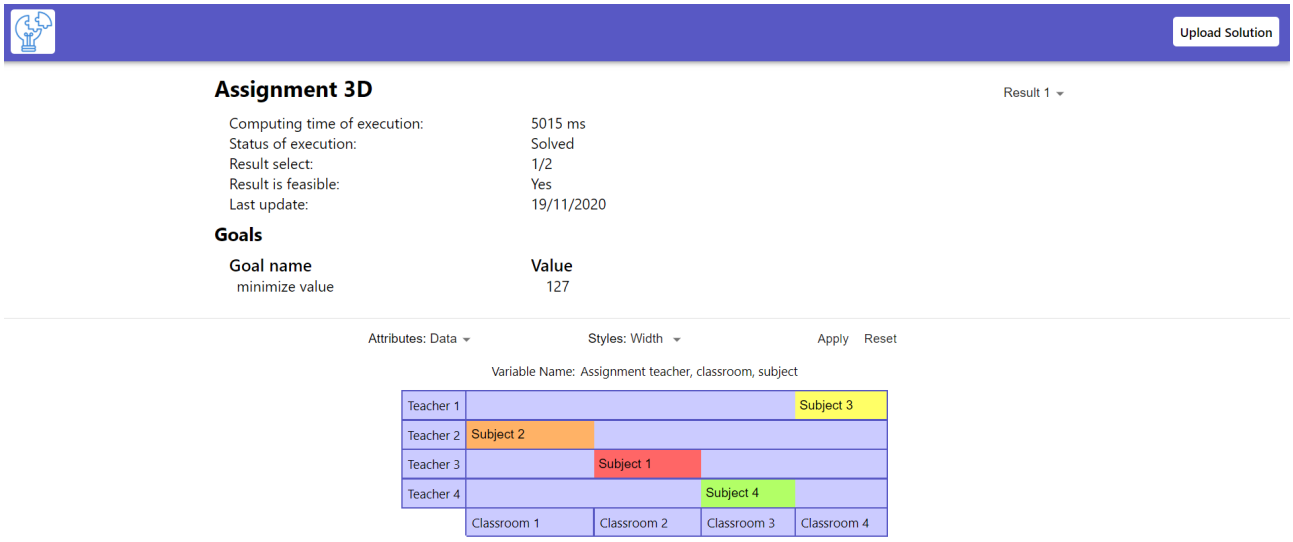


Figura 5.9: Problema de asignación 3D: resultado 1

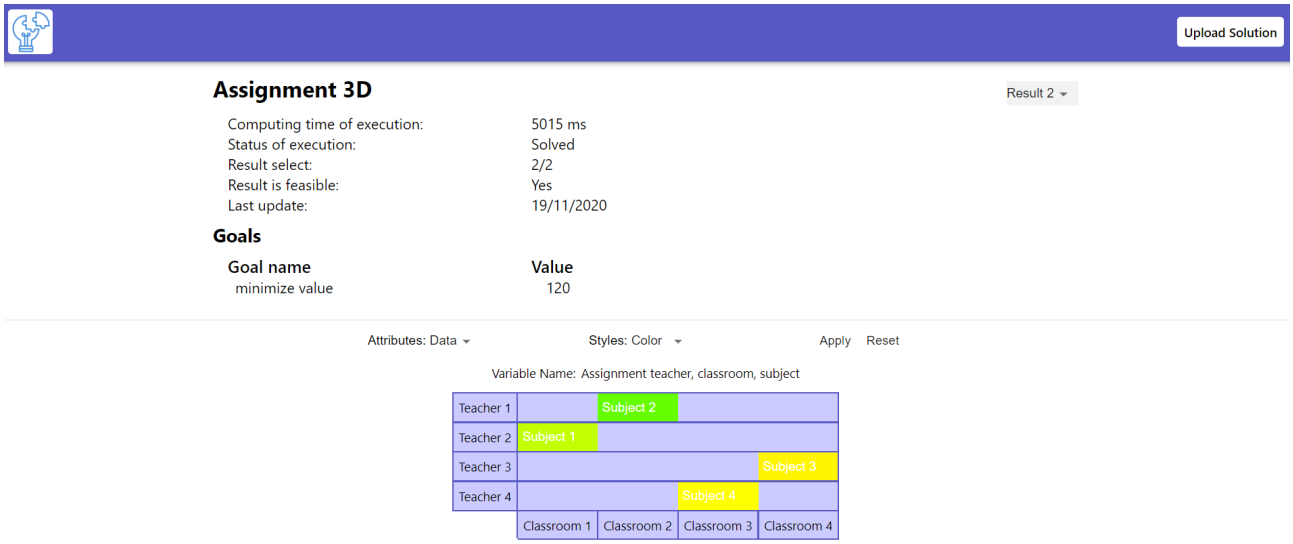


Figura 5.10: Problema de asignación 3D: resultado 2

Capítulo 6

Conclusiones y líneas futuras

Como hemos visto a lo largo de este Trabajo de Fin de Grado, la herramienta de Prodef tiene un alto nivel de complejidad integrada, el trabajo que tiene de fondo es digno de reconocer, durante su desarrollo se ha intentado que a simple vista para los usuarios sea fácil de manejar y comprender su uso. Para seguir con este objetivo, se ha desarrollado una interfaz para la visualización de soluciones, de forma tal que los usuarios sean capaz de comprender dichas soluciones y poder manipular su visualización, como se ha visto a lo largo de este trabajo.

Con el presente Trabajo de Fin de Grado, se ha logrado desarrollar una herramienta para la visualización de soluciones de Prodef, llamada Prodef-Solution. Con este proyecto se ha realizado:

- Comprensión de Prodef y ProdefLang a través de un estudio de la herramienta ya existente.
- Analizar los problemas y sus soluciones, obtenidos a través de la interfaz de Prodef.
- Representación visual de los resultados:
    - Problema de la mochila.
    - Problema de asignación 2D.
    - Problema del TSP.
    - Problema del VRP.
    - Problema de asignación 3D.
- Interacción con la visualización de los resultados, soportando 3 estilos:
    - Alto
    - Ancho
    - Color

Durante el desarrollo del proyecto, han surgido distintas dificultades, entre ellas destacar al principio del desarrollo que al intentar ejecutar el proyecto en el estado en el que lo encontré, me daba fallos, esto debido a que se encontraba en una versión de desarrollo. Otro dificultad que se encontró fue el cambio de la representación de la mochila para conseguir una mejor visualización, ya que con la primera representación, no se apreciaba bien la comparación de los resultados a la hora de cambiarlos.

Los principales objetivos del proyecto se han conseguido, pero es cierto que durante su elaboración se han encontrado puntos que abren paso a nuevas mejoras en la interfaz como:

- Agregar nuevos estilos para la visualización de los resultados.
- Mejorar la accesibilidad de la interfaz.
- Fusionar esta interfaz, con el proyecto de Prodef. Siendo esta la más importante.
- Permitir comparar dos resultados de un mismo problema en la misma ventana.
- Añadir nuevas representaciones a problemas que vayan surgiendo.

Capítulo 7

Summary and Conclusions

As we have seen throughout this Final Degree Project, the Prodef tool has a high level of integrated complexity, the work it has in the background is worthy of recognition, during its development it has been tried that at a glance for users easy to handle and understand its use. To continue with this objective, an interface has been developed for the visualization of solutions, in such a way that users are able to understand said solutions and be able to manipulate their representation, as has been seen throughout this work.

With this Final Degree Project, it has been possible to develop a tool for the visualization of Prodef solutions, called Prodef-Solution. With this project has been done:

- Comprehension of Prodef and ProdefLang through a study of the existing tool.
- Analyze the problems and their solutions, obtained through the Prodef interface.
- Visual representation of the results:
  - Knapsack problem.
  - 2D assignment problem.
  - TSP problem.
  - VRP problem.
  - 3D assignment problem.
- Interaction with the visualization of the results, supporting 3 styles:
  - Height
  - Width
  - Color

During the development of the project, different difficulties have surged, among them highlighting at the beginning of the development that when trying to execute the project in the state in which I found it, it gave me errors, this was due to the fact that it was in a development version. Another difficulty that was found was changing the representation of the Knapsack problem to achieve a better visualization, since with the first representation, the comparison of the results was not well appreciated when changing them.

The main objectives of the project have been achieved, but it is true that during its preparation points have been found that open the way to new improvements in the interface such as:

- Add new styles for displaying results.
- Improve the accessibility of the interface.
- Merge this interface with the Prodef project. This being the most important.
- Allow to compare two results of the same problem in the same window.
- Add new representations to problems as they arise.

Capítulo 8

Presupuesto

Un Trabajo de Fin de Grado tiene una asignación de 12 créditos, cada crédito son 25 horas. Por lo tanto, esto hace que está asignatura requiera de unas 300 horas de trabajo.

En la Tabla 8.1 podemos ver los costes de desarrollo. Para ello se ha realizado un desglose de las horas dedicadas a cada apartado o tarea de este proyecto.

Para el desarrollo se ha estimado un coste de 10€ por hora, partiendo de información publicada por https://www.infojobs.net/.

Titulo	Coste	Horas
Análisis de Prodef y de ProdefLang.	350	35
Análisis de problemas y sus soluciones.	700	70
Diseño e implementación de una interfaz gráfica para la representación de las soluciones.	1400	140
Validación y análisis de casos.	350	35
Documentación.	200	20
Total.	3000	300

Tabla 8.1: Costes de desarrollo

Apéndice A

Definición en Prodef

A.1. Problema de la Mochila


```
{
  "state": "resolved",
  "solution": {
    "results": [
      {
        "isFeasible": true,
        "goalValues": [
          {
            "expression": "sum x[i]*item[i].value over i=(1:N)",
            "sense": "maximize",
            "value": 130,
            "name": "Maximize the value of the items",
            "weight": 1
          },
          {
            "expression": "sum x[i]*item[i].value over i=(1:N)",
            "sense": "maximize",
            "value": 130,
            "name": "Maximize the value of the items",
            "weight": 1
          }
        ],
        "variableValues": [
          {
            "symbol": "x",
            "value": [1, 1, 1, 1, 0],
            "within": "integers",
            "name": "Items in the knapsack",
            "range": {
              "lowerBound": 0,
              "upperBound": 1
            },
            "shape": {
              "type": "vector",
              "isPermutation": false,
              "size": {
                "fixed": false,
                "value": "N"
              }
            }
```

```
        },
        {
          "symbol": "y",
          "value": [1, 1, 1, 1, 0],
          "within": "integers",
          "name": "Items in the knapsack",
          "range": {
            "lowerBound": 0,
            "upperBound": 1
          },
          "shape": {
            "type": "vector",
            "isPermutation": false,
            "size": {
              "fixed": false,
              "value": "N"
            }
          }
        }
      ]
    }
  ],
  "computingTime": 5015
},
"stateMessage": "The problem was successfully solved",
"problem": {
  "name": "Basic binary knapsack problem",
  "description": "Optional description (optimal: 130)",
  "parameters": [
    {
      "symbol": "N",
      "value": 5,
      "name": "Number of items"
    },
    {
      "symbol": "MaxWeight",
      "value": 80,
      "name": "Maximum weight"
    }
  ],
  "variables": [
    {
      "name": "Items in the knapsack",
      "shape": {
        "type": "vector",
        "isPermutation": false,
        "size": {
          "fixed": false,
          "value": "N"
        }
      },
      "symbol": "x",
      "within": "integers",
      "range": {
        "lowerBound": 0,
        "upperBound": 1
      }
    }
```

```
],
"goals": [
  {
    "name": "Maximize the value of the items",
    "expression": "sum x[i]*item[i].value over i=(1:N)",
    "sense": "maximize",
    "weight": 1
  }
],
"constraints": [
  {
    "expression": "sum x[i]*item[i].weight over i=(1:N) <= MaxWeight",
    "name": "Limit the total weight of the items in the knapsack"
  }
],
"classes": [
  {
    "attributes": [
      {
        "name": "Name",
        "symbol": "name"
      },
      {
        "name": "Value",
        "symbol": "value"
      },
      {
        "name": "Weight",
        "symbol": "weight"
      }
    ],
    "name": "Item",
    "symbol": "item"
  }
],
"objects": [
  {
    "attributes": [
      {
        "attribute": "name",
        "value": "Item 1"
      },
      {
        "attribute": "value",
        "value": 33
      },
      {
        "attribute": "weight",
        "value": 15
      }
    ],
    "class": "item"
  },
  {
    "attributes": [
      {
        "attribute": "name",
        "value": "Item 2"
```

```
    },
    {
      "attribute": "value",
      "value": 24
    },
    {
      "attribute": "weight",
      "value": 20
    }
  ],
  "class": "item"
},
{
  "attributes": [
    {
      "attribute": "name",
      "value": "Item 3"
    },
    {
      "attribute": "value",
      "value": 36
    },
    {
      "attribute": "weight",
      "value": 17
    }
  ],
  "class": "item"
},
{
  "attributes": [
    {
      "attribute": "name",
      "value": "Item 4"
    },
    {
      "attribute": "value",
      "value": 37
    },
    {
      "attribute": "weight",
      "value": 8
    }
  ],
  "class": "item"
},
{
  "attributes": [
    {
      "attribute": "name",
      "value": "Item 5"
    },
    {
      "attribute": "value",
      "value": 12
    },
    {
      "attribute": "weight",
```

```
            "value": 31
          }
        ],
        "class": "item"
      }
    ]
  },
  "lastUpdate": 1605879754703
}
```

Bibliografía

[1] Andrés Calimero García Pérez. Prodef: meta-modelado de problemas de optimización combinatoria. Technical report, Universidad de La Laguna, Julio 2020.

[2] Daniel González Expósito. Prodef-GUI: Interfaz gráfica para el modelado de problemas. Technical report, Universidad de La Laguna, Julio 2021.

[3] Blockly google developers. https://developers.google.com/blockly. Accessed: 2022-09-08.

[4] Google OrTools. https://developers.google.com/optimization/introduction/overview. Accessed: 2022-09-06.

[5] The Knapsack Problem. https://developers.google.com/optimization/bin/knapsack. Accessed: 2022-09-06.

[6] Solving an Assignment Problem. https://developers.google.com/optimization/assignment/assignm Accessed: 2022-09-06.

[7] Traveling Salesperson Problem. https://developers.google.com/optimization/routing/tsp. Accessed: 2022-09-06.

[8] Vehicle Routing Problem. https://developers.google.com/optimization/routing/vrp. Accessed: 2022-09-06.

[9] The Job Shop Problem. https://developers.google.com/optimization/scheduling/job_shop. Accessed: 2022-09-06.

[10] React. https://reactjs.org/. Accessed: 2022-09-06.

[11] Redux. https://es.redux.js.org/. Accessed: 2022-09-06.

[12] Material UI. https://mui.com/. Accessed: 2022-09-06.

[13] Salkin, Harvey M. and De Kluyver, Cornelis A. The knapsack problem: A survey. 1975.

[14] T. KACZOREK and M. SWIERKOSZ. Solution to the 2-D eigenvalue assignment problem. 1990.

[15] Yousefikhoshbakht, Majid. Solving the Traveling Salesman Problem: A Modified Metaheuristic Algorithm. 2021-02-19.

[16] Golden, Bruce L. and Wasil, Edward A. and Kelly, James P. and Chao, I-Ming. The Impact of Metaheuristics on Solving the Vehicle Routing Problem: Algorithms, Problem Sets, and Computational Results. 1998.

[17] Salvagnin, Domenico and Mittelmann, Hans. Solving a Challenging Quadratic 3D Assignment Problem