Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

PLANIFICACIÓN CLOUD BASADA EN UN SISTEMA FUZZY. ANÁLISIS DE INTERPRETABILIDAD

Alumno: Antonio Jiménez Sánchez

Tutor: Prof. D. José Enrique Muñoz Expósito
Depto.: Ingeniería de Telecomunicación

Noviembre, 2020

Índice General.

1. Resumen. 5

2. Introducción. 5

2.1. Contexto. 5

2.2. Estructura del documento. 6

3. Antecedentes. 6

3.1. Origen de la Lógica Difusa y evolución de su uso. 6

3.2. Trabajos anteriores. 7

4. Objetivos. 7

5. Materiales. 8

5.1. Lenguaje de programación, librerías y entorno de trabajo. 8

5.1.1. Python. 8

5.1.2. Librería Numpy. 9

5.1.3. Librería Flask. 9

5.1.3.1. Funciones de la librería Flask. 10

5.1.4. Librería Scikit-fuzzy. 10

5.1.4.1. Scikit-fuzzy control 10

5.1.5. Entorno Pycharm Community Edition. 11

5.2. Simulador WorkFlow. 11

5.2.1. Estrategia DVFS. 11

5.2.2. WorkflowSIM-DVFS. 12

5.3. Web Cloud Simulator User Interface. 13

5.3.1. Anotación inicial: Cambios realizados en la interfaz. 13

5.3.2. Vista principal. 14

5.3.3. Vista de selección de archivo de entrada. 14

5.3.4. Vista de selección del planificador. 15

5.3.5. Vista Datacenter. 15

5.3.6. Vista de virtualización. 16

5.3.7. Vista de simulación. 16

5.3.8. Vista de la lista de tareas. 18

5.3.9. Vista FAQ y “About us” 19

5.4. Docker. 19

5.4.1. Funcionamiento. 19

5.4.2. Creación de imágenes. 20
5.4.3. Instrucciones para la creación de un Dockerfile 20
5.4.4. Docker Compose. 21
5.4.5. Docker Machine. 21
5.4.6. Docker Swarm. 22
5.5. Conceptos teóricos. 22
5.5.1. Lógica Difusa. 23
5.5.1.1. Arquitectura general de la Lógica Difusa. 23
5.5.1.2. Ejemplo de Lógica Difusa. 23
5.5.2. Sistemas Basados en Reglas Difusas. 25
5.5.3. SBRD de tipo Mamdani. 26
5.5.3.1. Base de conocimiento de un SBRD de tipo Mamdani. 26
5.5.3.2. Bloque de inferencia de un SBRD de tipo Mamdani. 27
5.5.4. Ventajas y desventajas. 28
5.5.5. Variante del SBRD Mamdani: Mamdani con Forma Normal Disyuntiva. 28
5.6. Computación Evolutiva. 29
5.7. Algoritmos Evolutivos. 29
5.7.1. Estructura general de un algoritmo evolutivo. 29
5.8. Algoritmos genéticos. 30
5.8.1. Estructura general de un Algoritmo Genético. 30
5.8.2. Técnicas de recombinación y mutación. 32
5.9. Algoritmo genético de aprendizaje Pittsburgh. 34
5.9.1. La población de bases de reglas. 34
5.9.2. El sistema de evaluación. 34
5.9.3. El sistema de selección de individuos. 35
5.9.4. Sistema de aprendizaje genético. 35
5.10. Optimización por enjambre de partículas. 35
5.10.1. Estructura general de un PSO. 36
5.11. Algoritmo PSO KASIA. 38
5.11.1. Población. 38
5.11.2. Sistema de evaluación. 38
5.11.3. Actualización y mejora del enjambre. 38
5.12. Interpretabilidad de una base de reglas obtenida. 39

5.12.1. Sistema Difuso Jerárquico. 39
5.12.2. BR1 Dimensión de la base de reglas. 40
5.12.3. BR2 Complejidad. 41
5.12.4. BR 3Interpretabilidad. 41
5.12.5. BR 4 Índice de interpretabilidad. 42
6. Métodos. 43
6.1. Metodología de las aplicaciones desarrolladas. 43
6.1.1. Sistema de Inferencia Difusa. 43
6.1.2. Descripción de la base de datos. 43
6.1.3. Algoritmo genético Pittsburgh desarrollado. 45
6.1.3.1. Anotación inicial: Cambios introducidos. 45
6.1.3.2. Cambios en la representación de individuos. 45
6.1.3.3. Cambios en la integración del simulador. "WorkflowSim-DVFS". 47
6.1.3.4. Descripción global del algoritmo. 47
6.1.3.5. Definición de parámetros y creación de la población inicial. 48
6.1.3.6. Evaluación de individuos. 49
6.1.3.7. Mecanismo de selección. 50
6.1.3.8. Aprendizaje genético. 51
6.1.4. KASIA. 52
6.1.4.1. Configuración de parámetros y creación del enjambre. 53
6.1.4.2. Evaluación inicial del enjambre. 55
6.1.4.3. Bucle KASIA para la mejora y actualización del enjambre. 56
6.1.5. Descripción del Sistema Difuso Jerárquico desarrollado. 57
6.1.5.1. Intérprete de reglas. 57
6.1.5.2. Primer bloque. 58
6.1.5.3. Segundo bloque. 59
6.1.5.4. Tercer bloque. 60
6.1.5.5. Cuarto bloque. 61
6.1.6. Servicio web. 63
6.1.7. Integración en Docker. 65
6.1.7.1. Imagen de la base de datos y phpmyadmin. 65

6.1.7.2. Imagen del simulador. 65
6.1.7.3. Imagen del servicio web. 66
6.1.7.4. Manejo de la aplicación con Docker Compose. 66
6.1.7.5. Creación del clúster con Docker Swarm. 67
7. Resultados y discusión. 68
7.1. Resultados obtenidos y discusión para el cálculo del Índice de Interpretabilidad. 68
7.2. Resultados obtenidos y discusión para el algoritmo Pittsburgh. 70
7.3. Resultados obtenidos y discusión para el algoritmo KASIA. 72
7.4. Modificaciones realizadas en el simulador y discusión sobre las modificaciones. 73
7.5. Resultados obtenidos y discusión sobre el despliegue del clúster Docker. 74
8. Conclusiones. 78
9. Abreviaturas. 79
10. Referencias Bibliográficas. 80

1. Resumen.

Este proyecto realiza un análisis e implementación de sistemas y algoritmos que hacen uso de la Lógica Difusa (LD) en lenguaje Python. La LD funciona haciendo uso de reglas y conjuntos de pertenencia para obtener una salida en un sistema. Las reglas se generan utilizando la Computación Evolutiva (CE), que mediante simulaciones y algoritmos intenta solventar este inconveniente. Las reglas obtenidas con los métodos mencionados son procesadas por una aplicación intérprete desarrollada, que permite la obtención de parámetros como la interpretabilidad de las reglas, haciendo posible un análisis de estas.

Finalmente, se añade una aplicación web, el simulador "WorkflowSim" y se integran todos los elementos en tecnología Docker. La aplicación web permitirá la selección de parámetros de simulación, "WorkflowSim" generará resultados de consumo de energía para ser optimizados y Docker permitirá que el sistema desarrollado sea escalable y desplegable.

2. Introducción.

En el apartado de introducción se describe el contexto de este proyecto y la estructura seguida en el documento con una breve definición de cada capítulo.

2.1. Contexto.

En la actualidad, no es ningún secreto que el uso de la computación, junto con las diferentes ramas de informática, telecomunicaciones y electrónica ha contribuido con gran importancia al desarrollo tecnológico observado durante los últimos años. Sin embargo, con la ampliación de los campos de aplicación de la computación aumenta su uso, surgiendo una nueva serie de requisitos para las aplicaciones desarrolladas. Estos factores han incentivado la creación de redes de datos de alta velocidad, el aumento de capacidad de cómputo de las máquinas y la creación de una gran cantidad de aplicaciones y servicios, utilizados por muchos usuarios.

Debido a los hechos descritos, se genera e intercambia una gran cantidad de información y el requisito de almacenar estos datos en centros como los datacenters. No obstante, es tal la carga de trabajo que se requieren de mecanismos de ahorro de energía y métodos para automatizar y dar buscar a los problemas surgidos, propiciando el uso de algoritmos de aprendizaje y métodos de despliegue que faciliten estos servicios.

2.2. Estructura del documento.

- Resumen: este apartado realiza un análisis superficial de todos los elementos que se estudian y realizan a lo largo de este proyecto, junto a la relación existente entre ellos.
- Introducción: detalla la estructura del documento y el contexto existente durante realización de este.
- Antecedentes: indica los hechos que han llevado a la realización del proyecto presentado.
- Objetivos: el apartado de objetivos muestra las metas que se cubren con la realización del trabajo.
- Materiales: determina las herramientas que se han utilizado para la realización de todos los campos del proyecto.
- Métodos: especifica la metodología y técnicas utilizadas para lograr la realización del trabajo y cumplir los objetivos del proyecto.
- Resultados y discusión: esta sección recoge los resultados obtenidos tras la realización del proyecto y un análisis de su fiabilidad.
- Conclusiones: resoluciones finales sobre el trabajo realizado y los resultados de este.
- Abreviaturas: contiene todas las abreviaturas utilizadas a lo largo de este trabajo y el concepto al que sustituyen.
- Referencias bibliográficas: este apartado indicará todos los documentos que han sido utilizados para la creación de este proyecto.

3. Antecedentes.

A continuación, se muestran los antecedentes del proyecto para observar y comprender los hechos que han llevado a la realización de este trabajo.

3.1. Origen de la Lógica Difusa y evolución de su uso.

La computación utilizada tradicionalmente hace uso de la lógica booleana de ceros y unos. Esta implementación permite realizar una gran cantidad de operaciones de forma rápida y resolver problemas, pero puede resultar inefectiva si se aplica a problemas que no tengan soluciones exactas. Con esta premisa se buscó una forma de transmitir los

conocimientos a una máquina de una forma más similar al lenguaje humano, dando lugar a la implementación de la LD en estas.

Con el tiempo, los Sistemas Basados en Reglas Difusas (SBRD) cobraron importancia en el ámbito de aplicación la LD, caracterizando el sistema de reglas utilizado y demostrando ser efectivo en multitud de problemas. En concreto, el Sistema de tipo Mamdani fue el primero en tratar con entradas y salidas reales. Sin embargo, estos sistemas no tienen capacidad propia de generar conocimiento y mejorar, de modo que quedaron desfasados respecto a las necesidades de problemas más extensos y complejos, recurriendo a la CE para buscar un método de generación de conocimiento. Este esquema ha llegado hasta la actualidad.

3.2. Trabajos anteriores.

Este trabajo en particular también tiene como antecedente algunos componentes software desarrollados en el Trabajo de Fin de Máster titulado “Big data platform for the development of scheduling strategies in cloud computing”. En este trabajo, se realizó una aproximación al algoritmo genético de aprendizaje Pittsburgh y se creó una interfaz gráfica para seleccionar parámetros configurables. Las modificaciones, aportaciones y nuevas implementaciones realizadas en el presente Trabajo de Fin de Grado (TFG) se detallarán en los apartados correspondientes a lo largo de la memoria.

4. Objetivos.

Los objetivos de este proyecto se muestran a continuación:

- La comprensión del concepto de LD, su utilidad y los sistemas que hacen uso de esta.
- Realizar una introducción a la CE y algunas de sus distintas ramas como los Algoritmos Genéticos (AG).
- Comprender los algoritmos de aprendizaje automatizado, junto a los elementos necesarios para su implementación.
- Conocer el simulador “WorkflowSim”, un simulador que ayuda en la investigación de los flujos de trabajo de los centros de procesamiento de datos.
- Realizar un sistema capaz de medir la interpretabilidad de una serie de reglas.
- La realización de una aplicación web que integre los diferentes simuladores y algoritmos.

- Conocer los aspectos básicos de la tecnología Docker y su uso en aplicaciones para la creación de un clúster.

5. Materiales.

En el presente apartado se expondrán todas las herramientas, utilidades y conceptos usados durante el desarrollo de este proyecto, junto a una descripción y justificación de uso de cada elemento citado.

5.1. Lenguaje de programación, librerías y entorno de desarrollo.

Se indica en primer lugar el lenguaje utilizado, junto a las utilidades necesarias y el programa utilizado para trabajar con ambas partes dichas.

5.1.1. Python.

Durante el desarrollo de este proyecto se ha utilizado, principalmente, el lenguaje de programación Python.

Python es un lenguaje cuyo principal objetivo es proporcionar una sintaxis de programación que permita un código claro y descifrable. Este objetivo hace que el aprendizaje del lenguaje sea más sencillo y rápido que con otros lenguajes de programación.

Entre otros aspectos destacados en Python, se encuentra que este es un lenguaje de programación que:

- Se ejecuta sin un compilador que procese el código previamente. En caso de existir errores, estos se detectan en tiempo de ejecución.
- Permite hacer uso de múltiples paradigmas de programación existentes, como la programación orientada a objetos.
- Hace uso de tipado dinámica. Cuando se declara una variable, no es necesario decirle de que tipos son los datos [1].
- Es utilizable en múltiples sistemas operativos.
- Hace uso del código abierto las instrucciones de código de Python son gratuitas y se pueden ver o utilizar sin ninguna limitación.

Python también dispone de librerías y funciones que ayudan a realizar tareas que se repiten con frecuencia. Las librerías destacadas para este trabajo se describirán en los siguientes apartados.

5.1.2. Librería Numpy.

Numpy es una librería para dar apoyo a la creación de vectores y matrices multidimensionales en Python, junto a una gran variedad de funciones para realizar acciones sobre estos objetos [2]. Es muy utilizado en otros campos como la ciencia de datos.

Las funciones más comunes de esta librería son:

- Zeros: crea una matriz o vector de ceros en base a las dimensiones dadas.
- Ones: crea una matriz o vector de unos en base a las dimensiones dadas.
- Random: crea una matriz o vector relleno de número aleatorios en base a las dimensiones dadas. Se puede definir el rango en el que deben situarse dichos números aleatorios.
- Where: busca un elemento en la matriz en base a una condición.
- Min: busca el valor mínimo que se encuentra dentro de una matriz o vector dado.
- Max: busca el valor mínimo que se encuentra dentro de una matriz o vector dado.
- Floor: redondea los valores contenidos en el vector o matriz correspondiente hacia el entero más bajo.
- Round: redondea los valores contenidos en un elemento vector o matriz al número de decimales indicado.
- Size: proporciona el número total de valores que componen el vector dado.
- Arange: crea un vector de valores numéricos con una separación equidistante entre los valores adyacentes.

5.1.3. Librería Flask.

Flask es un entorno de trabajo ligero para la creación de aplicaciones web, diseñado para ser sencillo de implementar y permitir que las aplicaciones sean escalables [3]. Cuenta con un servidor web de desarrollo integrado, por lo que no es necesario contar con una infraestructura previa, y un depurador en tiempo de ejecución para observar cambios realizados en el código.

En cuanto a estructura de aplicación, el entorno Flask se organiza de forma predeterminada en los siguientes directorios:

- “Templates”: este directorio debe de guardar los ficheros web a renderizar para ser vistos en el navegador web cuando se acceda a ellos desde la aplicación realizada.

- “Static”: es un directorio que almacena archivos estáticos, que no varían entre sesiones, como las imágenes o archivos de hojas de estilo.
- Directorio raíz: contendrá los archivos de código programados en Python con el entorno Flask. Estos archivos accederán a las otras carpetas y especificarán las rutas y vistas a mostrar, entre otras funciones.

5.1.3.1. Funciones de la librería Flask.

Las funciones más importantes de la librería Flask son:

- Route: especifica una URL para acceder a un recurso. Cualquier ruta debe ir seguida de una función escrita en Python, que será la encargada de manejar la petición realizada en la aplicación.
- Render_template: función utilizada para que se muestre una vista concreta de la aplicación. De forma predeterminada, buscará la vista en el directorio “templates”.
- Methods: indica si la petición realizada es de tipo “get” o “post”.
- Config: permite configurar parámetros opcionales, como la recarga automática de ciertos archivos.
- Run: función encargada de poner en marcha la aplicación. Debe ir siempre al final del fichero que implemente la aplicación. Dentro de este método, se deben añadir siempre una serie de parámetros de gran importancia como:
    - “Debug”: indica si se activa o no el depurador integrado en la librería.
    - “Host”: indica la dirección IP en la que se aloja el servicio.
    - “Port”: puerto que utilizará la aplicación.

5.1.4. Librería scikit-fuzzy.

Esta librería proporciona muchas funciones y herramientas útiles para la computación en proyectos que involucran la LD. La mayoría de las funciones se alojan en sub-paquetes, como el módulo ‘control’ [4].

5.1.4.1. Scikit-fuzzy-control.

Proporciona una interfaz de programación de aplicaciones para el diseño de sistemas difusos. Para ello proporciona las siguientes funciones [5]:

- “Antecedent”: define una variable antecedente para un sistema difuso.
- “Consecuent”: define una variable consecuente para un sistema difuso.

- “Rule”: define las reglas para un sistema difuso. Estas reglas conectarán posteriormente las variables antecedentes con las variables consecuentes que se hayan definido.
- “ControlSystem”: implementa el sistema de control difuso. Recibe una serie de reglas para implementar el sistema.
- “ControlSystemSimulation”: realiza el cálculo de resultados una vez se ha definido el sistema de control citado anteriormente.

5.1.5. Entorno Pycharm Community Edition.

Pycharm es un entorno de desarrollo utilizado para la creación de aplicaciones en lenguaje Python. Proporciona capacidades de análisis de código, depuración, localización y corrección de errores, integración de consola del sistema operativo y soporte para el desarrollo web. [6]

5.2. Simulador Workflow.

El crecimiento de las capacidades de los sistemas de computación en la nube (llamados comúnmente sistemas “cloud”) depende, en gran medida, de la reducción del consumo de energía de los datacenters que formen parte del sistema para hacerlos sostenibles y económicos [7]. Este tipo de servicios generan a su vez cargas de trabajo denominadas “workflows”, cuya optimización en cuanto al consumo de energía se encuentra en un estado poco avanzado. Para la mejora de la gestión de energía de estos flujos de trabajo existen diferentes utilidades, como el simulador ‘Workflow Sim’ o la estrategia “Dynamic Voltage Frequency Scaling” (DVFS).

El simulador utilizado, denominado “WorkflowSim-DVFS” por el uso de dicha estrategia, está diseñado para funcionar en multitud de escenarios mejorando el uso de los recursos propios de cada máquina del datacenter ofreciendo a su vez soluciones realistas.

5.2. Estrategia ‘Dynamic Voltage Frequency Scaling’.

La estrategia DVFS permite el ajuste de energía y velocidad de los diversos elementos de una máquina, como la CPU, para optimizar los recursos asignados a cada proceso y así mejorar el ahorro de energía y el tiempo de vida de dicha máquina.

DVFS puede ejecutarse a nivel interno (mejorando la eficiencia dentro de la propia máquina que lo ejecute) o a nivel de red (mejorando la comunicación entre máquinas, estableciendo criterios de coordinación y cooperación). En este caso se ejecutará a nivel

interno, ya que la integración de la estrategia DVFS al simulador “WorkflowSim” es relevante para permitir la gestión de energía de una máquina anfitrión del Datacenter mediante el rendimiento dinámico de la CPU [7].

Para este caso particular, se debe atender al uso de DVFS en el Kernel de linux, donde se pueden encontrar distintos modelos que gestionan las operaciones y requisitos de funcionamiento del sistema denominados reguladores que dirigen las técnicas DVFS. Existen 5 tipos de reguladores DVFS [17].

- Rendimiento: establece dos límites de frecuencia fijos, denominados “frecuencia mínima de escalado” y “frecuencia máxima de escalado”. El gobernador establecerá la frecuencia de funcionamiento en el límite superior.
- Ahorro de energía: igual que la técnica “rendimiento”, pero esta vez la frecuencia de funcionamiento se situará en el límite inferior.
- Espacio de usuario: establece la frecuencia según un programa utilizado por el usuario.
- Bajo demanda: realiza un ajuste de frecuencia en base al uso de la CPU y el límite de uso predefinido.
- Conservador: realiza un ajuste de la frecuencia basado en el uso de la CPU, pero de una forma más gradual.

5.2.2. WorkflowSim-DVFS.

En el simulador encontramos 5 entidades principales para la gestión de la carga de trabajo:

1. El planificador: inicia el sistema y analiza un archivo DAX para obtener tareas de forma individual. Estos archivos DAX son flujos de trabajo en forma de grafo directo (sin bucles) escritos en formato XML.
2. Concentrador: recibe las tareas desde el planificador y agrupa las diferentes tareas en conjuntos denominados trabajos.
3. Motor: recibe los trabajos determinados por el concentrador y asegura que se ejecutan en el orden correcto.
4. Planificador: selecciona la máquina más adecuada para realizar estos trabajos en el momento determinado.
5. La máquina del Datacenter que realizará el proceso.

5.3. Web Cloud Simulator User Interface.

La aplicación “Web Cloud Simulator User Interface” (WCS-UI) tiene como principal objetivo proporcionar una interfaz al usuario para la configuración del simulador “Workflow” y las aplicaciones que se realicen. A lo largo de este apartado se indicará cada una de las vistas que componen la interfaz, junto con los diferentes cambios aplicados en ellas respecto a su versión inicial.

5.3.1. Anotación inicial: Cambios realizados en la interfaz WCS-UI.

La mayor parte de esta interfaz no ha sido realizada por el autor de este proyecto. Sin embargo, la elaboración de la aplicación web escrita en Python que hace uso de la interfaz, junto a otra serie de cambios, sí ha sido totalmente realizada en este proyecto. Los cambios se detallan a continuación:

- Se ha realizado una aplicación web, escrita totalmente en lenguaje Python, que permite la integración de la interfaz de usuario con los algoritmos y aplicaciones desarrollados a lo largo del proyecto.
- Los archivos que componen la interfaz se han estructurado de acuerdo con la librería descrita en el apartado 5.1.3. Este hecho ha supuesto cambiar la ruta de acceso a diversos archivos en el código de las vistas. Estos cambios no alteran la apariencia inicial de la aplicación.
- Respecto a la vista que muestra la lista de tareas, descrita en el apartado 5.3.8 de este documento, se han añadido diferentes funcionalidades como:
    - El acceso a una tabla de resultados en una nueva pestaña del navegador web que se utilice.
    - La descarga de las reglas obtenidas en una simulación cualquier desde la propia página. Estas reglas se descargan en un archivo de texto plano, almacenándose en la base de datos especificada y en archivos locales dentro del directorio de la aplicación realizada.
    - La actualización automática de la vista, de modo que sea posible la observación de los cambios realizados a lo largo de las simulaciones, así como el estado en el que se encuentran estas.
    - La limpieza de la memoria caché cada vez que se inicia la aplicación web.

5.3.2. Vista principal.

Una vez se ha accedido a la interfaz de la aplicación mediante el navegador web, se mostrará la vista principal de la aplicación. Esta página permite acceder al resto de vistas existentes en la aplicación web. Estas vistas se describen de forma inicial a continuación:

- Vista de selección de archivo de entrada: permite seleccionar el archivo que utilizará el simulador "Workflow".
- Vista de selección de planificador: indica el planificador utilizado para la simulación.
- Vista de configuración de datacenter: selecciona la configuración de las máquinas del datacenter.
- Vista de virtualización: establece parámetros para la creación de máquinas virtuales.
- Vista de simulación: permite seleccionar el tipo de simulación y el algoritmo utilizado.
- Vista de visualización de tareas: esta vista permite el acceso a la tabla de resultados almacenados sobre las simulaciones realizadas.
- Vistas de explicación de uso e información: permiten el acceso a información sobre la interfaz y la organización dónde se realizó.

La figura 1 muestra la apariencia de la vista principal.

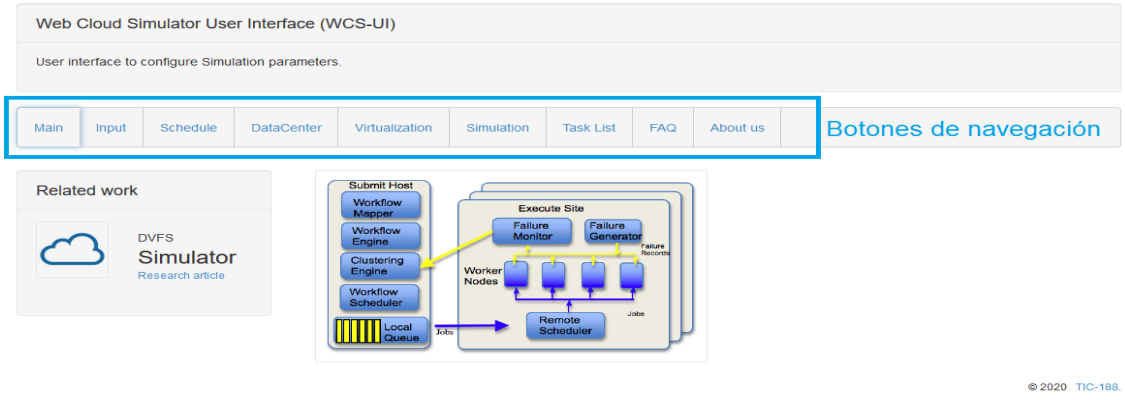

Figura 1. Vista principal de WCS-UI.

5.3.3. Vista de selección de archivo de entrada.

En caso de acceder al botón "Input", se mostrará un menú desplegable en el cual se podrá seleccionar el archivo DAX de entrada al simulador "WorkflowSim-DVFS". El archivo seleccionado, ya sea de tipo "inspiral", "montage" o "sipht" se utilizará para la futura simulación.

La figura 2 permite ver cómo es el menú desplegable de la vista.

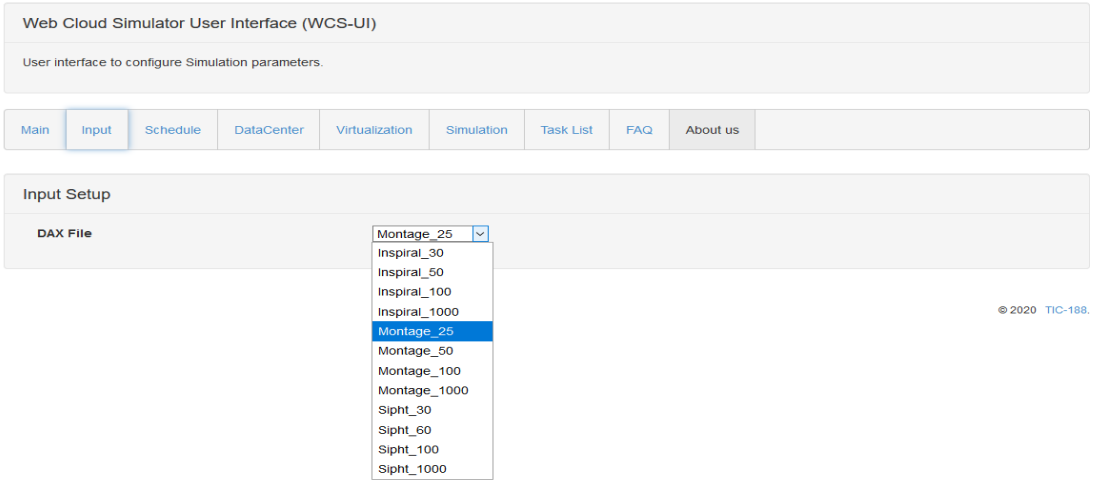

Figura 2. Vista de selección de archivo de entrada de WCS-UI

5.3.4. Vista de selección de planificador.

Esta vista ofrece otro menú desplegable desde el que se podría definir el comportamiento del planificador de tareas utilizado en el simulador "WorkflowSim" durante su ejecución. La figura 3 permite ver cómo es la vista descrita.

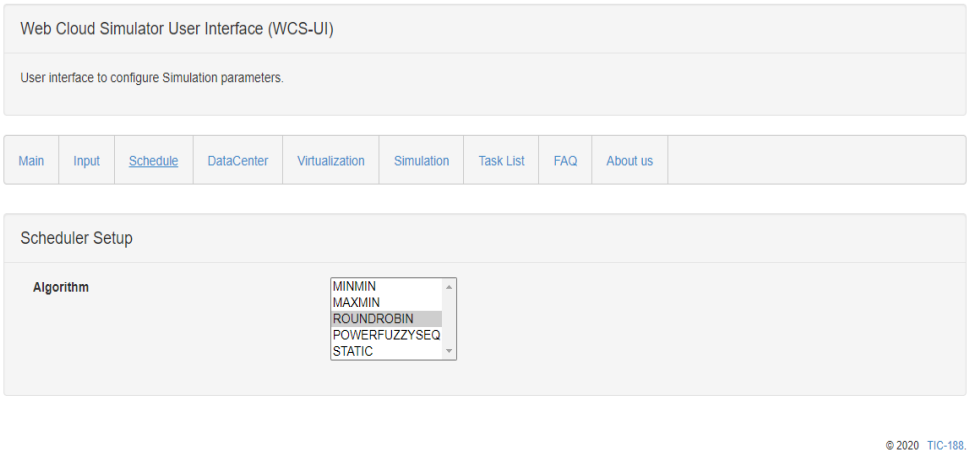

Figura 3. Vista de selección de planificador de WCS-UI

5.3.5. Vista datacenter.

La vista datacenter de la aplicación web permite establecer características de las máquinas que componen el Datacenter, como el sistema operativo utilizado, la arquitectura del procesador, el número de hosts existentes o la disciplina utilizada para la técnica DVFS.

En la figura 4 se puede observar cómo es la vista datacenter.

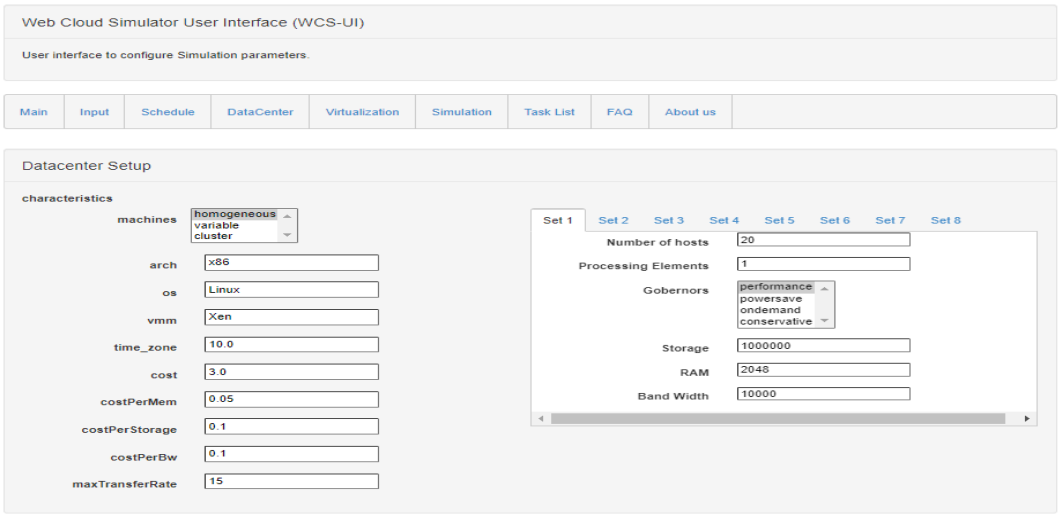

Figura 4. Vista datacenter de WCS-UI.

5.3.6. Vista de virtualización.

Permite la configuración de las máquinas virtuales de cada host, estableciendo parámetros como la memoria RAM, los millones de instrucciones realizados por segundo o el administrador de máquinas virtuales.

La vista descrita se muestra en la figura 5.

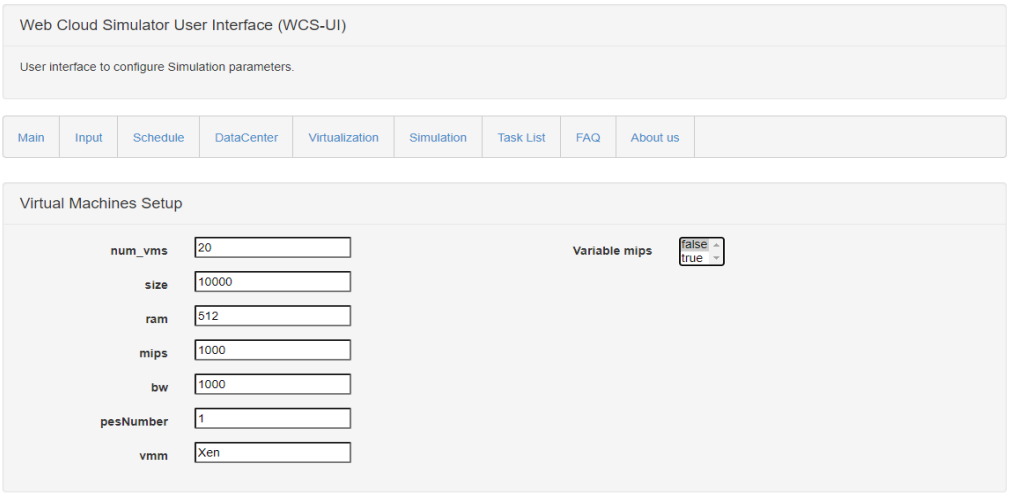

Figura 5. Vista de virtualización de WCS-UI.

5.3.7. Vista de simulación.

En este apartado se configuran parámetros referidos a la simulación que se realizará. Entre estos parámetros se pueden encontrar algunos como el tipo de simulación o el nombre que recibe para su almacenamiento en una base de datos.

Pueden observarse 2 vistas posibles, ya que hay 2 tipos de simulaciones:

- Simulación de entrenamiento: realiza una simulación para la obtención de reglas. Requiere de parámetros como el algoritmo de aprendizaje a utilizar, el patrón (por ejemplo, tiempo o consumo de energía) a optimizar o el número de iteraciones. En

la figura 6, presentada a continuación, se puede mostrar la interfaz mostrada para este caso.

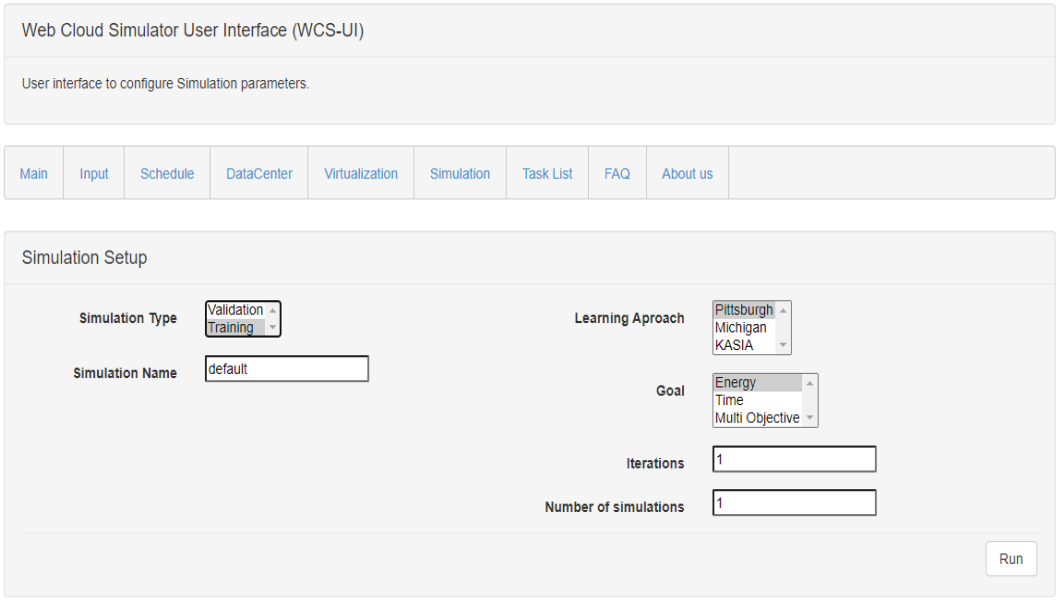

Figura 6. Vista para una simulación de tipo entrenamiento.

- Simulación de validación: la validación utilizará unas reglas obtenidas con algún procedimiento realizado previamente, como una simulación de tipo entrenamiento. Las reglas estarán almacenadas en un fichero y se podrán añadir desde la propia interfaz, especificando la ruta. La figura 7 muestra la vista simulación en el caso de seleccionar el tipo validación.

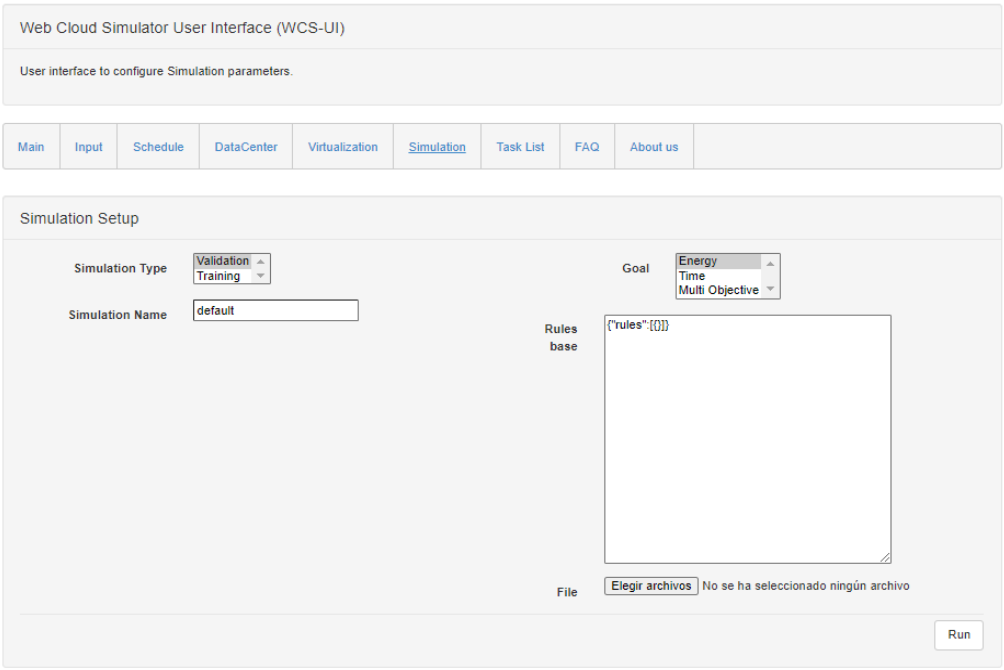

Figura 7. Vista para una simulación de tipo validación.

5.3.8. Vista de la lista de tareas.

En esta parte de la aplicación web hay un botón denominado “View all” que, en caso de ser utilizado, mostrará todas las simulaciones existentes en la base de datos independientemente del estado en el que se encuentren.

La figura 8 muestra la vista y el botón nombrado.

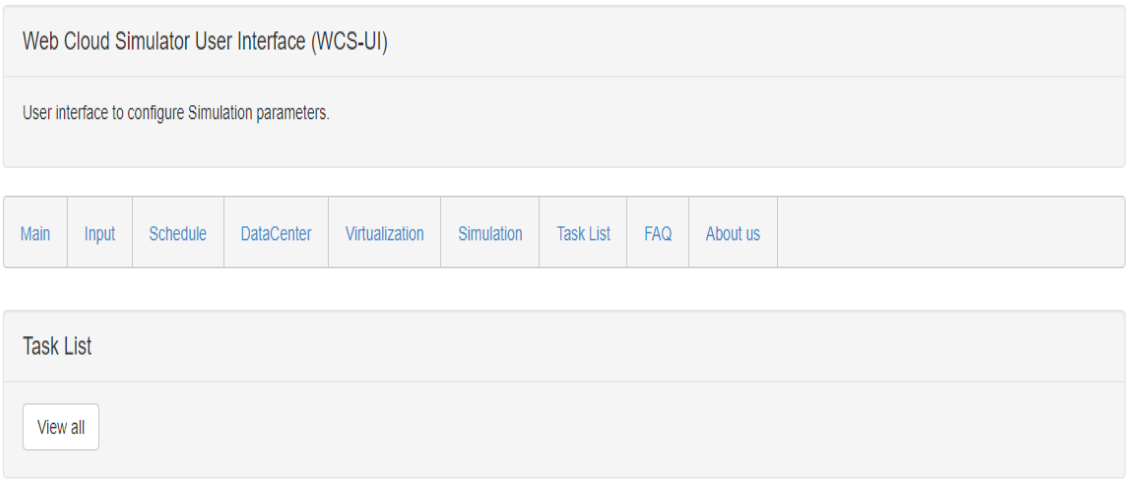

Figura 8. Vista de lista de tareas.

En caso de pulsar el botón, se accederá a la vista mostrada en la figura 9. A continuación, se hacen algunas reflexiones sobre esta vista:

- Las simulaciones pueden estar finalizadas o en proceso, por lo que la nueva pestaña generada en el navegador al utilizar el botón cuenta con una actualización automática para mantener actualizada la información de la simulación.
- Se puede acceder a la descarga de las reglas generadas o utilizadas en la simulación en un archivo de texto. Para ello, se debe pulsar en el tipo de algoritmo de aprendizaje utilizado.

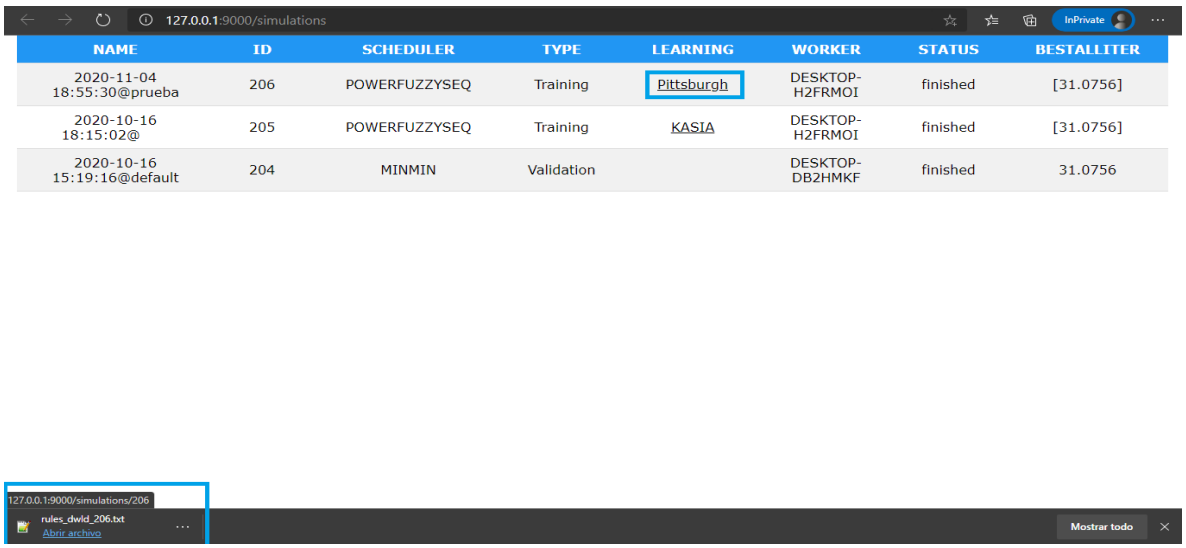

Figura 9. Vista de lista de tareas.

5.3.9. Vista de explicación de uso e información.

En caso de acceder a la pestaña FAQ, se podrá leer una descripción de la interfaz y las ventajas de su uso. El botón About us, simplemente redirige a la página web de la Universidad de Jaén. Esta vista se muestra en la figura 10.

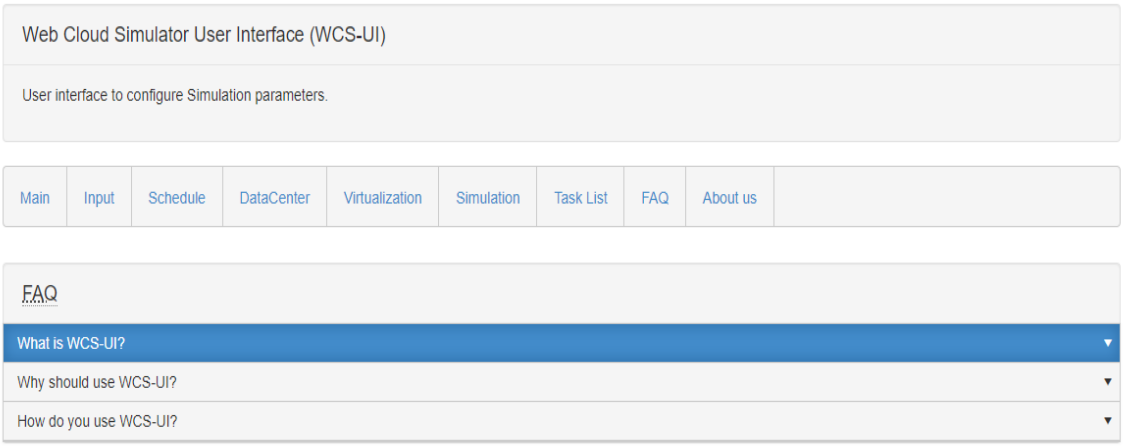

Figura 10. Vista de lista de tareas.

5.4. Docker.

Docker es una tecnología de virtualización ligera que permite la creación de paquetes de software sin necesidad de virtualizar la capa del sistema operativo. Estos paquetes se denominan contenedores y contendrán todos los elementos necesarios (bibliotecas, código, requisitos de la aplicación, etc) para que las aplicaciones se puedan ejecutar en cualquier máquina independientemente de su sistema operativo, por lo que su despliegue es más sencillo, escalable y consume menor cantidad de recursos como el uso de memoria RAM y CPU.

5.4.1. Funcionamiento.

La tecnología Docker hace uso del kernel de Linux y permite la separación de los procesos necesarios para que se ejecuten de forma independiente, mejorando aspectos como la seguridad al considerarse sistemas separados [11]. Aunque la tecnología se basó en contenedores linux tradicionales, actualmente se ha desmarcado de estos para ofrecer mayores capacidades.

La implementación en Docker se realiza mediante imágenes, utilizadas para crear los contenedores. Estas imágenes son representaciones de sus respectivas aplicaciones junto a su configuración y dependencias. Debido a esto, las imágenes son bastante ligeras, permitiendo una sencilla distribución de las aplicaciones, control de versiones y la mecanización de los procesos de la aplicación por parte del propio Docker.

El uso de las imágenes comentadas en Docker se realiza mediante comandos. Algunos de los comandos más utilizados son:

- “docker run”: procede a la ejecución de un contenedor.
- “docker ps”: realiza un listado de todos los contenedores que hay funcionando.
- “docker inspect id”: proporciona una gran cantidad de información sobre el contenedor especificado.
- “networks settings”: proporciona información sobre la red en la que operan los contenedores.
- “docker ps -a”: muestra todos los contenedores existentes.
- “docker stop id”: este es un comando que procede a la detención del contenedor especificado.
- “docker start id inicia”: este comando procede a la iniciación o reanudación de un contenedor específico.
- “docker attach id”: permite la conexión directa a la salida del contenedor mediante consola por parte del usuario.

5.4.2. Creación de imágenes.

Docker permite crear imágenes propias de una aplicación, ya sea para integrar en Docker la aplicación local o modificar una externa haciendo uso de los denominados “Dockerfiles”. Los Dockerfile son archivos de texto compuestos por las estructuras e instrucciones requeridas para formar una imagen y junto al comando “docker build” [14], permite formar una imagen indicando el Dockerfile y los archivos que forman la aplicación en la ruta local especificada.

5.4.3. Instrucciones para la creación de un Dockerfile.

Docker ejecutará las instrucciones del Dockerfile línea a línea. Las instrucciones con las que crear un Dockerfile son [13]:

- FROM: Todo Dockerfile debe comenzar con esta instrucción, que indica la imagen desde la que se comienza a construir la aplicación.
- RUN: la instrucción run ejecuta el comando seleccionado y asignará los resultados de la ejecución.
- CMD: esta instrucción tiene como propósito proporcionar un valor predeterminado para un contenedor en ejecución, es decir, ejecuta el archivo principal de la aplicación en el contenedor. Solo puede existir un comando CMD.

- LABEL: añade metadatos a la imagen, como la versión en la que esta se encuentra. Una etiqueta se compone por una clave y su valor. Puede existir más de una etiqueta.
- ENV: establece una variable de entorno para la construcción de la imagen.
- MAINTAINER: indica el autor de la imagen. Sin embargo, Docker recomienda el uso de LABEL por ser más flexible.
- COPY: copia archivos o directorios desde el sistema local al contenedor en la ruta especificada.
- WORKDIR: establece el directorio de trabajo para los comandos anteriores y así encontrar archivos auxiliares de la aplicación.

5.4.4. Docker Compose.

Docker Compose es una utilidad para indicar a Docker cómo se debe levantar la aplicación desarrollada, para que así sea capaz de dirigir los diferentes servicios que la componen. Su uso se basa en la creación y utilización de archivos YAML. Entre sus características destaca la posibilidad de reconstruir aplicaciones en caso de alguna modificación. [15]

El uso de Docker Compose facilita mucho el despliegue de la aplicación, pues hace posible ejecutar multitud de comandos que permiten:

- Iniciar, reconstruir y parar cualquier servicio que forme parte de la aplicación en caso de ser necesario.
- Observar el estado de los servicios en ejecución.
- Observar los registros de salida

En cuanto a la utilización de Docker Compose, esta se realiza mediante los siguientes pasos:

1. Definir el entorno de la aplicación mediante un Dockerfile.
2. Definir los servicios de la aplicación en un archivo denominado “docker-compose.yml”.
3. Ejecutar el comando docker-compose up para construir la aplicación.

5.4.5. Docker machine.

Docker Machine es una herramienta que permite la creación y configuración de máquinas, tanto virtuales como físicas, con la tecnología Docker implementada haciendo uso de comandos. Esta gestión mediante comandos permite a su vez un sistema centralizado, aunque se haga uso de sistema remotos.

Las capacidades más importantes de Docker Machine son:

- La capacidad de crear máquinas en entornos locales como VirtualBox o sistemas en la nube.
- Aporta seguridad al sistema utilizando certificados TLS o el protocolo SSH para una comunicación segura entre las máquinas creadas.

5.4.6. Docker Swarm.

Las capacidades de orquestación y administración de clústeres en Docker se realizan utilizando "swarmkit". Este es un proyecto separado que implementa la capa de orquestación de Docker y se usa directamente dentro de él.

Un enjambre Docker consiste en múltiples host que implementan la tecnología Docker y se ejecutan en modo enjambre ("swarm mode"). Los diferentes nodos pueden actuar como administradores, gestionando la pertenencia de un modo al enjambre y delegando funciones, y como trabajadores, ejecutando cualquier servicio indicado por el administrador. Un host puede desempeñar ambos modos. [16]

Cuando se crea un servicio, se define su estado óptimo, proporcionando parámetos como el número de réplicas del servicio, la red en la que actúan, puertos, etc. Esta herramienta intenta mantener este estado deseado y por ello, si un nodo trabajador deja de estar disponible, la propia tecnología Docker administra las tareas de ese nodo a los otros nodos. Una tarea en un contenedor en funcionamiento que forma parte del enjambre, gestionada por el administrador. [16]

Una de las ventajas clave de los servicios en enjambre es la posibilidad de modificar la configuración de un servicio, sin necesidad de reiniciar el servicio. El propio funcionamiento de Docker swarm, actualizará la configuración y creará nuevos nodos automáticamente.

5.5. Conceptos teóricos.

En este apartado se procede a definir todos los conceptos teóricos necesarios para la realización y comprensión de este trabajo. En los siguientes apartados se definirán elementos como la LD, los sistemas que hacen uso de reglas y distintos algoritmos evolutivos que generan reglas para estos sistemas.

5.5.1. Lógica Difusa.

La LD es un tipo de lógica basada en grupos de pertenencia aplicados junto a las variables de un problema, a diferencia de la lógica tradicional de verdadero o falso utilizada en la computación.

Tiene su origen en el problema de la comprensión del lenguaje natural o humano por parte de los computadores y por ello, el uso de la LD ofrece soluciones y modos de comportamiento aproximados en vez de exactos, similares al lenguaje natural utilizado por los humanos, lo que permite su aplicación a problemas no realizables mediante el uso de la computación binaria, ya que muchos fenómenos no son exactos en la realidad. [9]

5.5.1.1. Arquitectura general de la Lógica Difusa.

De forma general, La LD se compone de los elementos mostrados en la figura 11 y funciona de la siguiente forma:

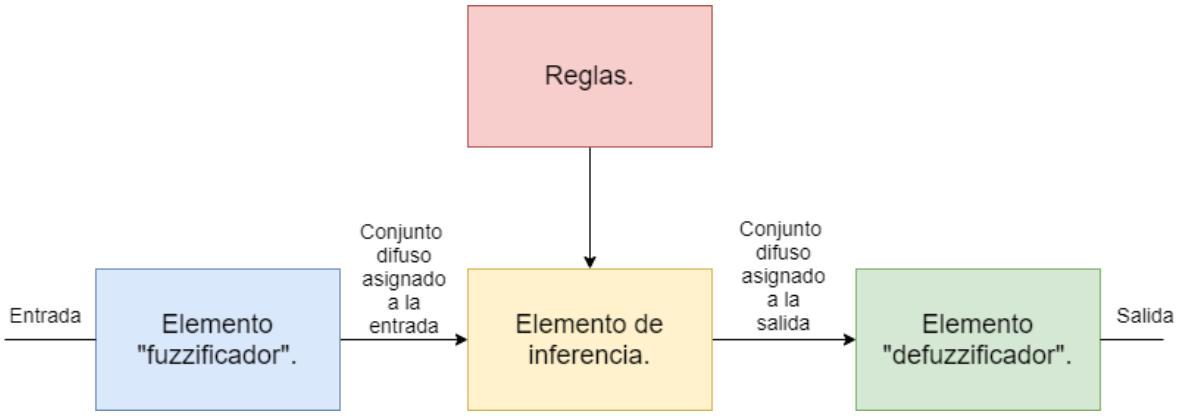

Figura 11. Componentes de la Lógica Difusa.

1. En primer lugar, se deben definir los conjuntos difusos de entrada y salida junto a las reglas que serán utilizadas para el proceso de inferencia.
2. Se transforma una entrada al sistema en el conjunto difuso de entrada correspondiente.
3. Se realiza el proceso de inferencia, determinando un conjunto fuzzy de salida al respectivo conjunto fuzzy de entrada junto a la información dada.
4. Finalmente, el conjunto borroso de salida obtenido del proceso de inferencia se transforma a una salida.

5.5.1.2. Ejemplo de Lógica Difusa.

En este apartado se expone un ejemplo de LD compuesto por 2 entradas y 1 salida relacionados mediante una sola regla:

- La primera entrada considerada es el precio de un producto, compuesta por los conjuntos difusos de pertenencia “bajo”, “medio” y “alto” representados con las funciones “Z de Riemann”, “Gaussiana” y “Sigmoide” respectivamente.
  Su representación se muestra en la figura 12.

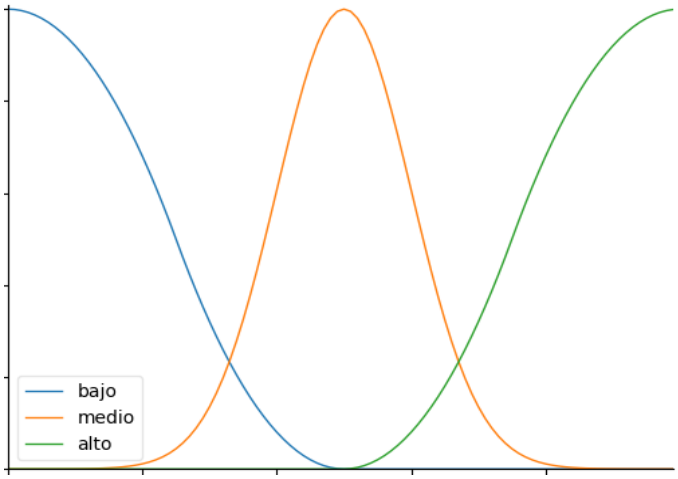

Figura 12. Conjuntos de pertenencia de la variable “precio”.

- La segunda entrada considerada es el material utilizado, compuesta también por 3 conjuntos difusos de pertenencia llamados “malo”, “medio” y “bueno". Se han representado con las funciones “Z de Riemann”, “Gaussiana” y “Sigmoide” respectivamente.
  La figura 13 muestra los conjuntos de pertenencia difusa de la variable descrita.

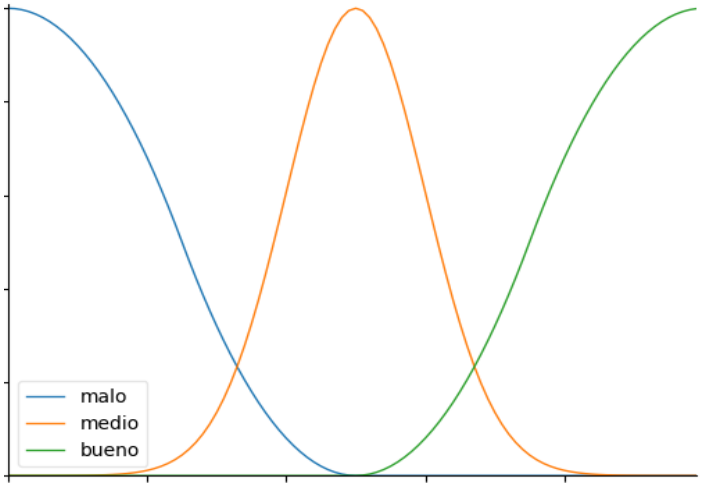

Figura 13. Conjuntos de pertenencia de la variable “material utilizado”.

- La salida es la calidad del producto, en la que encontramos 3 grupos difusos de pertenencia. Estos conjuntos se han realizado mediante funciones triangulares, observables en la figura 14.

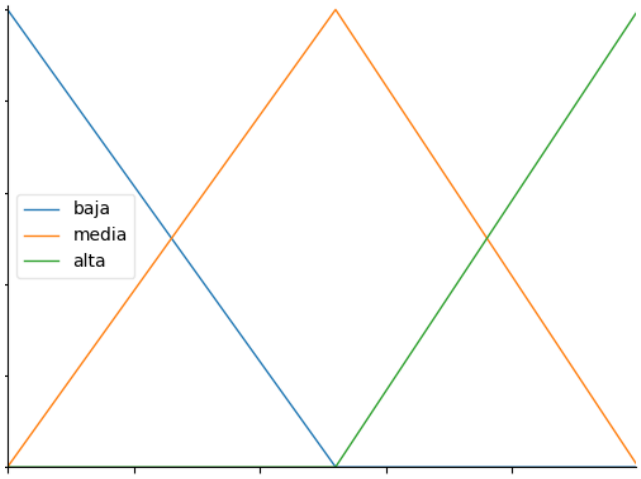

Figura 14. Conjuntos de pertenencia de la variable “calidad”.

- Se define la regla a utilizar para este ejemplo de LD: “Si el precio es alto y el material es bueno, la calidad del producto es alta”.
- En este punto, los antecedentes y consecuentes están descritos y unidos por reglas. La figura 15 muestra en color verde los conjuntos de pertenencia que entran en acción para este ejemplo. Las variables se ubicarán en estos conjuntos, pero por el momento, no se establece el valor que toman estas variables, pues se detallará en apartados posteriores.

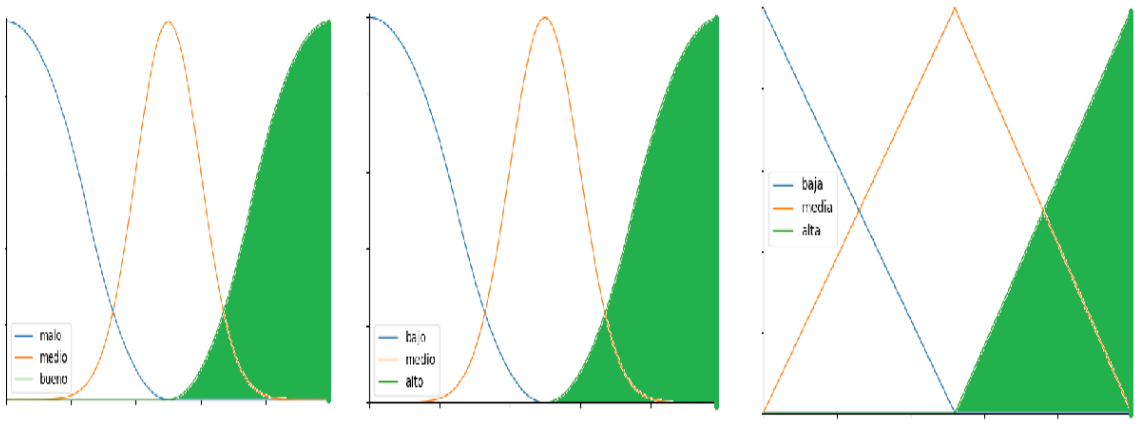

Figura 15. Conjuntos de pertenencia activados por la regla descrita.

5.5.2. Sistemas Basados en Reglas Difusas.

Los sistemas basados en reglas han tenido éxito modelando problemas humanos y comportamientos adaptativos, representando el conocimiento humano como reglas condicionales.

Un Sistema Basado en Reglas Difusas (SBRD) es un sistema basado en reglas donde la LD es utilizada como herramienta para representar distintas formas de conocimiento sobre

un problema, así como las interacciones y relaciones entre las diferentes variables. Esta propiedad permite que sea aplicado a una gran cantidad de problemas de diferentes tipos.

Los SBRDs comprender reglas ‘sí-entonces’ cuyos antecedentes y consecuentes se comprenden de enunciados difusos, presentando ventajas principales sobre el sistema clásico basado en reglas:

- Las reglas difusas implican incertidumbre y aproximación, 2 características importantes y frecuentes en este ámbito.
- Los métodos de inferencia se vuelven más robustos y flexibles.
- La representación del conocimiento es más sencilla gracias al uso de variables lingüísticas.

Para problemas dentro del ámbito de la ingeniería, existen distintos tipos de SBRD, siendo uno de los más utilizados es el SBRD de tipo Mamdani.

5.5.3. SBRD de tipo Mamdani.

Este tipo de SBRD trabaja con entradas y salidas del sistema reales, haciendo uso de reglas totalmente lingüísticas y una serie de conjuntos borrosos para caracterizar y catalogar las entradas y salidas. Entre sus características encontramos que este sistema difuso hace uso de interfaces de “fuzzificación” y “defuzzificación”, junto al uso de un razonamiento muy similar al visto en el apartado de LD. [9]

La figura muestra la estructura básica de un SBRD de tipo Mamdani. Sus componentes se detallarán en los siguientes apartados.

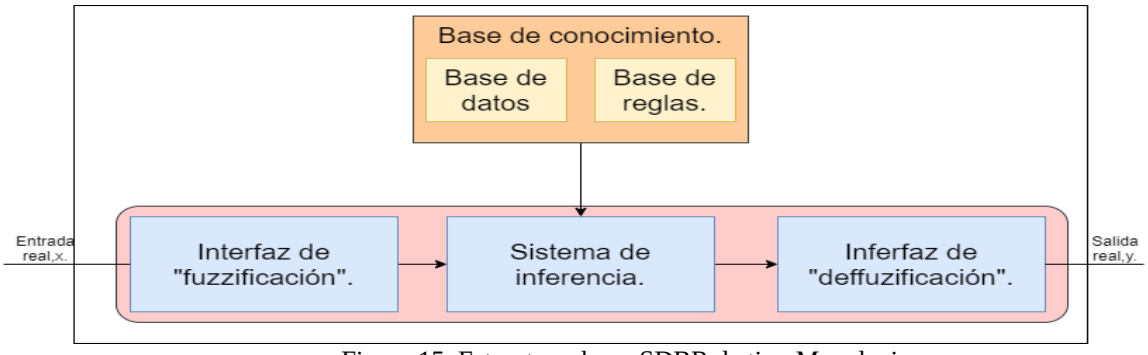

Figura 15. Estructura de un SDBR de tipo Mamdani.

5.5.3.1. Base de conocimiento de un SBRD de tipo Mamdani.

La Base de Conocimiento (BC) es la parte fundamental de un SBRD de tipo Mamdani. Sirve como almacén del conocimiento que se tiene sobre un problema específico, modelando la relación entre la entrada y la salida de los sistemas subyacentes, relación sobre la cual se apoya razonamiento que realiza el proceso de inferencia.

La estructura de las reglas de este sistema debe de ser lo más genérica posible para que sea aplicable a multitud de problemas. La estructura más utilizada posee la siguiente forma:

"Si X_1 es A_1 y ... X_i es A_i , entonces W es B"

donde:

- X y W son los valores de entrada y salida del sistema respectivamente.
- A y B son etiquetas lingüísticas con un grupo borroso asociado. Por ejemplo, una entrada sería el precio de un producto y su etiqueta "caro" o "barato".
- El subíndice i es un número entero que hace referencia al número de entradas existentes para el sistema.

Esto provoca que la base de reglas tenga distintos niveles de información, la definición de un conjunto borroso y las reglas lingüísticas:

- Las etiquetas y las funciones de pertenencia a un grupo borroso, que se encuentran en la Base de Datos (BD).
- Las reglas lingüísticas se corresponden con la Base de Reglas (BR). Estas reglas comprenden los conjuntos difusos definidos unidos mediante operadores lógicos, como 'and' y 'or'. A su vez, las reglas pueden estar formadas por uno o varios conjuntos de pertenencia difusa y múltiples reglas podrían activarse para una misma entrada.

5.5.3.2. El bloque de inferencia de un SBRD de tipo Mamdani.

El sistema de inferencia de un SBRD de tipo Mamdani está compuesto por tres componentes:

- Una interfaz de "fuzzificación": componente encargado de transformar un valor de entrada real al sistema en un conjunto de pertenencia difusa. Para dicho proceso, se utilizará la información almacenada en la BX, como los conjuntos difusos y las reglas, por lo que se convierte un valor preciso es un dato impreciso.

  Se utilizan diferentes funciones matemáticas como la Gaussiana, triangular o trapezoidal entre otras para la creación de distintos conjuntos difusos.
- Un sistema de inferencia: componente que deduce desde una entrada difusa hacia distintos conjuntos difusos de salida de acuerdo con la información que haya almacenada en la KB.

- Una interfaz de “defuzzificación”: convierte el conjunto difuso obtenido en el proceso de inferencia a un valor real de salida que constituye la salida global del SBRD. Es más complejo de implementar debido a que realiza una conversión de un dato impreciso a uno preciso.

5.5.4. Ventajas y desventajas.

Este apartado indica algunas ventajas y desventajas del uso del sistema de tipo Mamdani:

- Forma sencilla de incluir conocimiento mediante reglas lingüísticas. Permite una interpretación sencilla por parte de las personas.
- Alto grado de libertad para seleccionar el grupo de fuzzificación, defuzzificación y método de inferencia, por lo que el sistema se puede adaptar al problema.
- Cuando las variables de entrada son dependientes entre sí, puede resultar complejo encontrar un grupo borroso de salida.
- Poca flexibilidad en el sistema debido a la definición rígida de los conjuntos borrosos.
- El tamaño de la BC crece rápidamente con el número de variables y reglas en el sistema. Además, a mayor número de información, mayor complejidad para el proceso visto, por lo que no es un sistema escalable.

5.5.5. Variantes del SBRD Mamdani: Disjunctive Normal Form Mamdani.

Debido a las desventajas contempladas en el apartado anterior, surge una variante del SBRDde tipo Mamdani de igual estructura, pero con unas reglas que otorguen mayor flexibilidad de cara al problema.

Las nuevas reglas estarán compuestas por grupos de antecedentes formados por el operador lógico ‘or’ y estos grupos a su vez estarán unidos por el operador ‘and’. La nueva estructura que tendrían estas reglas se muestra en el enunciado inferior:

“Si X_1 es $\{A_{1a}$ o $A_{1b}\}$ y ... Xi es $\{A_{ic}$ o $A_{id}\}$ Entonces W es B_e”

donde:

- X y W son los valores de entrada y salida del sistema respectivamente.
- A y B son etiquetas lingüísticas con un grupo borroso asociado.
- El subíndice i es un número entero que hace referencia al número de entradas existentes para el sistema.
- Las letras a,b,c,d y e son números enteros cualesquiera.

5.6. Computación evolutiva.

La CE es el conjunto de técnicas para la solución de problemas de búsqueda y aprendizaje basadas en la teoría de la evolución biológica. Dentro de estas técnicas se encuentran los algoritmos evolutivos.

5.7. Algoritmos evolutivos.

Los algoritmos evolutivos constituyen una forma de búsqueda y optimización que imitan la evolución natural. Engloban diferentes ramas de investigación como los algoritmos genéticos, estrategias evolutivas, programación evolutiva y programación genética.

Su principal modo de operación se basa en conceptos genéticos, una población con soluciones candidatas y combinaciones aleatorias o alteración entre las soluciones para generar nuevas soluciones. A este proceso, se le añade un mecanismo de selección que descarte ciertas soluciones y seleccione las mejores.

5.7.1. Estructura general de un algoritmo evolutivo.

La estructura general de un algoritmo evolutivo se compone de los siguientes pasos:

1. En primer lugar, se debe generar la población que forme parte del algoritmo evolutivo y que será objeto de evaluación y mejora.
2. Se debe evaluar la población generada anteriormente y otorgar a cada miembro un tipo de puntuación que ayude a diferenciar qué individuos son mejores que otros.
3. Se seleccionan los mejores individuos existentes en la población para aplicar distintas técnicas sobre ellos que generen nuevos descendientes.
4. Se recombinan los mejores individuos, mezclando características de estos entre sí para formar la nueva descendencia.
5. Aplicación de técnicas de mutación, modificando alguna característica de los individuos de forma que se amplíe el espacio de búsqueda de soluciones del algoritmo y mantener la diversidad dentro de la población.
6. Se crea una nueva población con los descendientes generados.
7. Se comprueba si se ha alcanzado el criterio de terminación. En caso afirmativo finaliza el algoritmo evolutivo. En caso negativo, se debe volver a evaluar la nueva población que hayamos generado y completar las siguientes acciones hasta alcanzar el criterio.

La figura 16 muestra de forma esquemática la estructura descrita:

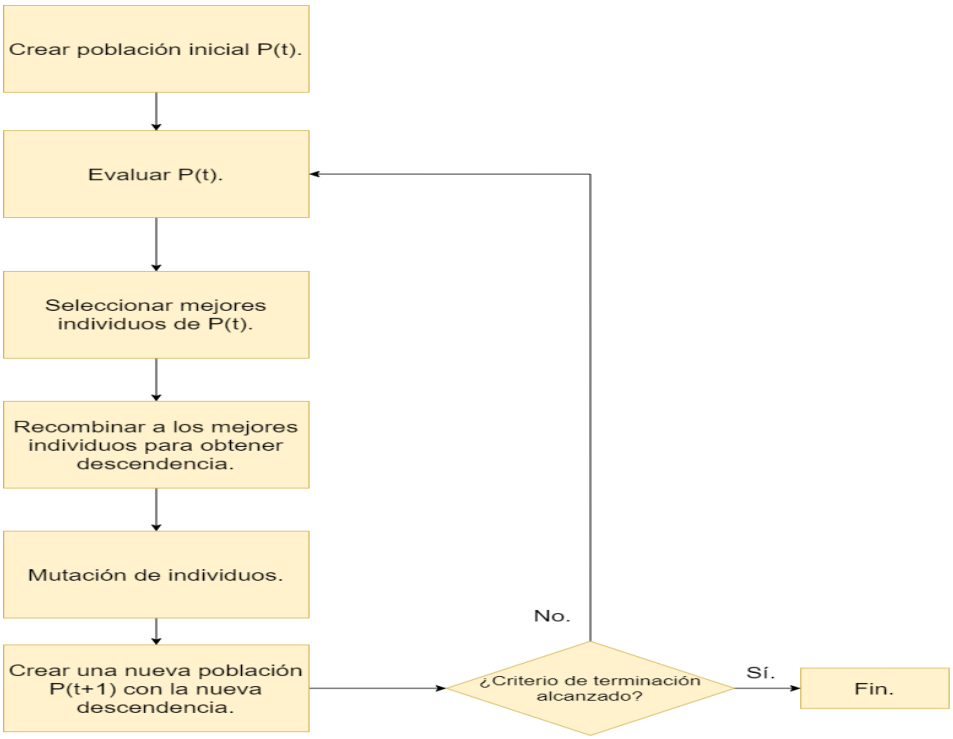

Figura 16. Estructura general de un algoritmo evolutivo.

5.8. Algoritmos genéticos.

Los AG son un tipo de algoritmo evolutivo de búsqueda, de modo que la estructura de estos algoritmos genéticos es similar a la de los algoritmos evolutivos, inspirado por la genética y la selección natural para hacer evolucionar soluciones a un problema. Proveen un sistema robusto de búsqueda en espacios complejos, así como una aproximación válida a problemas que requieran de búsquedas eficientes y efectivas.

Este algoritmo opera sobre una población de soluciones candidatas aleatorias o que, en caso de no ser aleatorias, sean bastante diferentes entre sí para que el algoritmo no converja rápidamente. La población avanza hacia mejores soluciones aplicando operadores genéticos en cada paso creando nuevas generaciones. En cada iteración, el mejor individuo encontrado en una población se irá aproximando a la solución óptima.

5.8.1. Estructura general de un algoritmo genético.

1. Se genera un conjunto de individuos llamado población, donde cada uno de estos es una posible solución. Cada individuo tendrá variables denominadas genes y un conjunto de genes formará un cromosoma, de modo que el cromosoma caracteriza por completo al individuo. Generalmente, los genes y cromosomas se representan de forma vectorial utilizando valores binarios, pero dicha representación no se restringe solo a esta norma. A este proceso se denomina también codificación en cromosomas.

Para mayor interpretabilidad, en la figura 17 se expone un ejemplo gráfico.

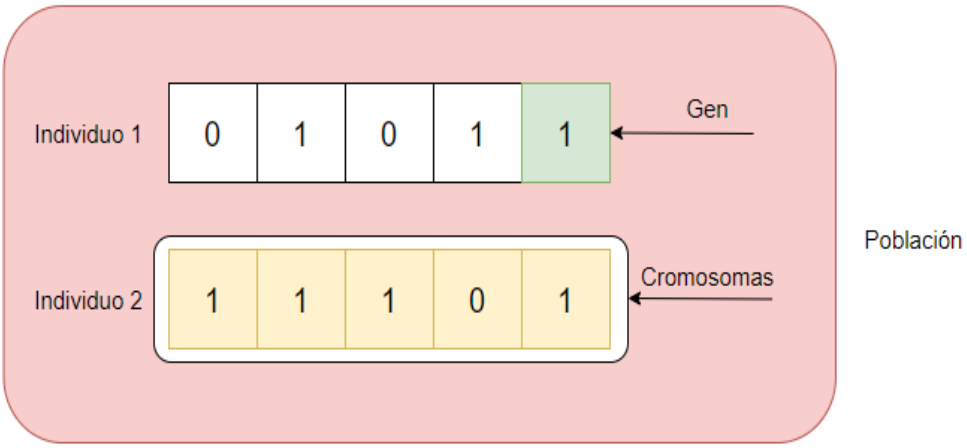

Figura 17. Ejemplo de codificación en cromosomas de un individuo.

2. Se evalúa a los individuos, otorgándole una puntuación de calidad a cada individuo para posteriores operaciones.
3. Se aplican diferentes operadores genéticos para que se produzca la evolución hacia la solución. Los operadores genéticos aplicados son:
    a. Reglas de selección: Selecciona a los mejores individuos dentro de la población en base a la puntuación de calidad obtenida durante la evaluación.
    b. Recombinación: Los mejores individuos serán combinados entre sí, creando nuevas y mejores soluciones para la siguiente población.
    c. Reglas de mutación: Se aplican aleatoriamente a individuos, modificándolos para crear descendencia y mantener la diversidad dentro de la población. La mutación también es una herramienta útil para descubrir zonas aún no analizadas y evitar que el algoritmo converja rápidamente
4. Se comprueba si se alcanzaron los criterios de optimización. Pueden darse 2 casos diferentes:
    a. Se han alcanzado los criterios: en este caso se deduce que se ha alcanzado la solución óptima y se toma dicha solución de la población, finalizando el algoritmo genético.
    b. No se han alcanzado los criterios, de modo que se debe evaluar de nuevo la población y aplicar los operadores genéticos durante una nueva iteración del proceso.

La estructura definida se muestra en la figura 18:

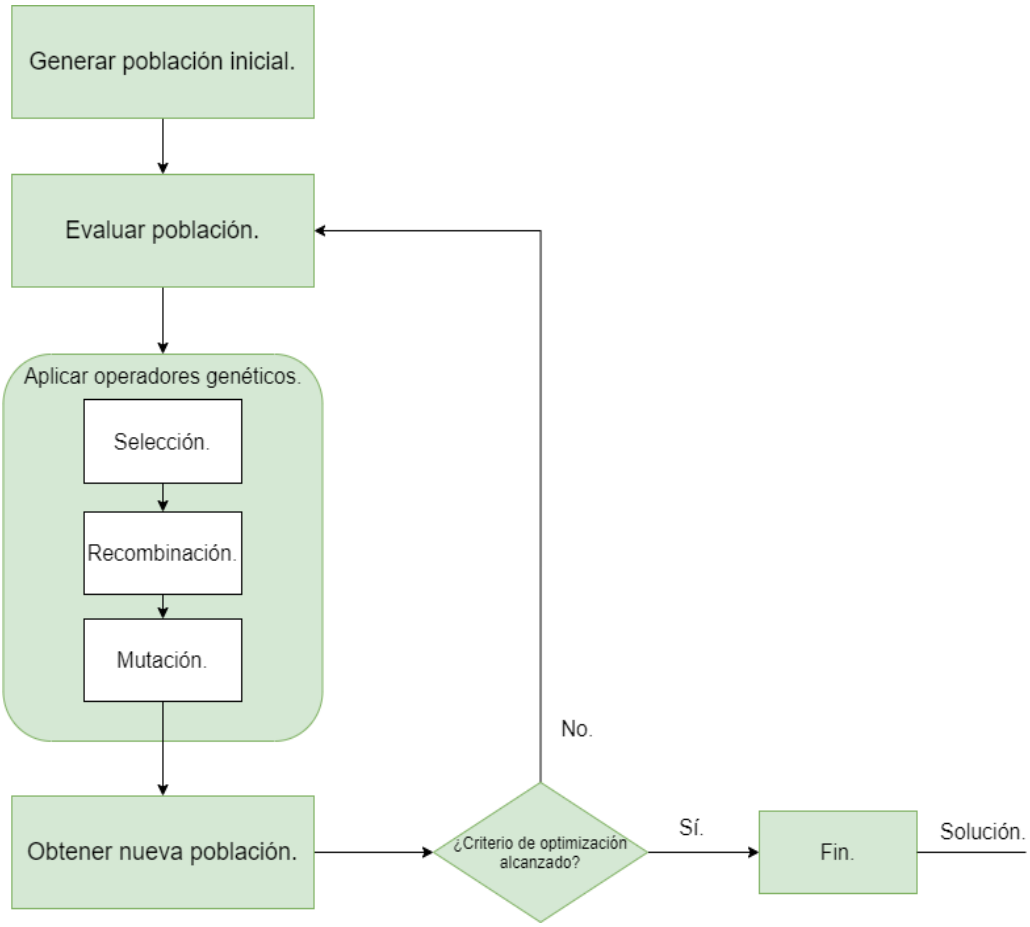

Figura 18. Estructura general de un algoritmo genético.

5.8.2. Técnicas de recombinación y mutación.

- Recombinación de un sólo punto: Se sitúa un punto de corte en los cromosomas de dos individuos, intercambian los segmentos resultantes tras el punto entre estos individuos. El punto de corte debe estar en la misma posición en ambos cromosomas para obtener un segmento de igual longitud.

  En la figura 19 se puede apreciar una implementación de esta técnica:

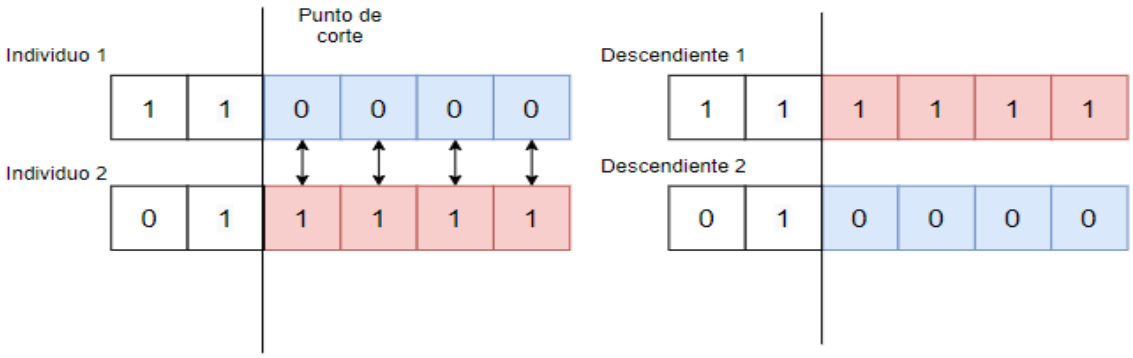

Figura 19. Recombinación de un solo punto.

- Recombinación de dos puntos: se sitúan dos puntos de corte en los cromosomas de dos individuos de modo que los segmentos de cromosoma existentes entre esos dos puntos cambien al otro individuo. Al igual que en el caso anterior los puntos de corte deben estar en la misma posición en ambos cromosomas.

  La figura 20 muestra un ejemplo de esta técnica:

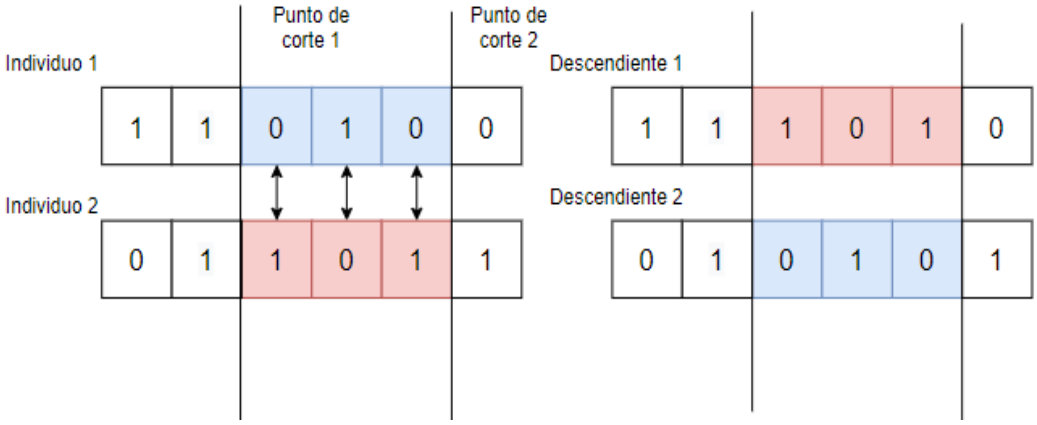

Figura 20. Recombinación de dos puntos.

- Mutación por probabilidad: Los genes de un individuo tienen una probabilidad de cambiar antes de finalizar cada iteración.
- Corte y unión: Los cromosomas de los individuos de la población elegidos como padres se cortan por una posición aleatoria, generando diferentes segmentos. Estos segmentos se unirán luego para formar descendientes. También es importante destacar que esta técnica se puede usar tanto para la recombinación como para mutación, pues los segmentos de un mismo individuo se pueden colocar en un orden diferente. [9]

  Se expone un ejemplo visual en la figura 21:

Figura 21. Técnica de corte y unión.

5.9. Algoritmo de Aprendizaje Genético Pittsburgh.

Pittsburgh es un algoritmo adaptado para el entrenamiento en multitud de problemas, cercano a la idea de evolución mediante competición y adaptación de individuos al entorno.

Cada individuo es una entidad de conocimiento y por ello las interacciones entre individuos están fuera de la evaluación de ese conocimiento, de modo que la medida de capacidad se aplica directamente a cada individuo en solitario. Los individuos son BR, de modo que la población está formada por bases de reglas que se evalúan de forma interna en el sistema de una en una para posteriormente, aplicar operadores que permitan crear descendientes con los mejores individuos encontrados.

Para realizar la función de sistema de aprendizaje, Pittsburgh cuenta con 4 elementos principales:

- La población de BR.
- El sistema de evaluación.
- El sistema de selección de individuos.
- El sistema de aprendizaje genético.

5.9.1. La población de bases de reglas.

El proceso de aprendizaje trabaja sobre una población de BR donde cada BR es una posible solución al problema. Esta población generalmente estará generada aleatoriamente, pero pueden existir casos donde exista conocimiento previo disponible para crearla.

Cada BR es evaluada de forma independiente y sin interacción con otras BR, aplicándola al problema en cuestión y observando el comportamiento de esta en el sistema.

5.9.2. El sistema de evaluación.

La evaluación de una BR se realiza mediante el efecto que tiene la aplicación de esta en el sistema y la respuesta del entorno.

El sistema de evaluación suele ser el elemento que más tiempo necesita en el proceso, debido a que la evaluación aislada de cada individuo reduce el número de interacciones e intercambio de información entre individuos, pero aumenta la carga de trabajo por el alto número de evaluaciones a realizar.

5.9.3. El sistema de selección de individuos.

La función de este sistema es la selección y ordenación de los individuos para colocar las soluciones más aptas en la zona de elitismo y permitir que el bloque de aprendizaje genético trabaje con los mejores candidatos.

5.9.4. El sistema de aprendizaje genético.

La función del sistema de aprendizaje genético es la generación de nuevas poblaciones de BRs aplicando operadores genéticos como la mutación o la recombinación a la población actual, una vez que todos los individuos de esta hayan sido evaluados, con su posterior selección.

Por otro lado, se debe tener en cuenta factores como los porcentajes de elitismo (qué número de mejores soluciones se seleccionan y permanecen para la siguiente población) y eliminación de individuos (qué proporción de la población abandonará para dejar paso a nuevos individuos obtenidos). El establecimiento de valores inapropiados podría derivar en una convergencia prematura del proceso al no existir una variedad suficientemente amplia entre las características de la población.

5.10. Optimización por enjambre de partículas.

La Optimización por Enjambre de Partículas (PSO) es un método de optimización basado en el comportamiento de ciertas especies (como un banco de peces o una banda de pájaros) con el fin de encontrar mínimos o máximos globales durante diferentes iteraciones, intentando obtener así la solución óptima.

A diferencia de los algoritmos genéticos, los PSO son más sencillos de implementar, requiere de menos parámetros fijos y las partículas o posibles soluciones se actualizan sin necesidad de operadores genéticos como la mutación o la combinación entre partículas. Además, logran una rápida convergencia sin ser muy sensibles al tamaño de la población, lo que permite que sea un método de optimización escalable.

Las partículas de un PSO tienen memoria e intercambios de información unidireccional reduciendo el número de comunicaciones entre partículas. Para lograr esta tarea, cada partícula debe estar definida por 4 elementos:

- Posición: el estado en que se encuentra la partícula en cada iteración, utilizada para evaluar la calidad de la partícula.

- Velocidad: la dirección y magnitud de movimiento de la partícula en cada iteración.
- La mejor posición que haya logrado la partícula.
- La mejor posición que haya logrado el enjambre.

5.10.1. Estructura general de un PSO.

La estructura general de un PSO es la siguiente:

1. Se crea un grupo de partículas aleatorias para formar la población inicial. Cuando se genera el enjambre de partículas, cada partícula sólo conoce su posición y su velocidad. Sin embargo, a lo largo del proceso cada partícula debe de almacenar 2 importantes valores para su posterior evolución.
    a. La mejor posición que ha alcanzado la partícula, denominado PBest.
    b. La mejor posición que ha alcanzado cualquier partícula del enjambre, llamado Gbest.
2. Se evalúa Pbest en cada partícula y en caso de haber obtenido un valor mejorado, se actualiza Pbest.
3. En caso de existir variación en Pbest, se debe comparar el nuevo Pbest con la mejor posición global Gbest almacenada. En caso de que el nuevo valor mejore a Gbest, se actualizará Gbest.
4. Se calcula la nueva velocidad para la partícula utilizando los valores Pbest, Gbest y diversos parámetros junto a la fórmula de la ecuación 1.

$$V(t+1) = W * V(t) + D1 * R1 * [Pbest(t) - V(t)] + D2 * R2 * [Gbest(t) - V(t)] \quad \text{(Ecuación 1)}$$

donde:

- V (t+1) es la nueva velocidad.
- W es un coeficiente de inercia.
- V (t) es la velocidad actual de la partícula.
- D1 es un valor de conocimiento referido al enjambre.
- R1 es un elemento aleatorio con valores entre 0 y 1 e igual dimensión la velocidad.
- Pbest (t) es la mejor posición que ha alcanzado la partícula.
- D2 es un valor social referido al enjambre.

- R2 es un elemento aleatorio con valores entre 0 y 1 e igual dimensión la velocidad.
- Gbest (t) es la mejor solución que ha alcanzado cualquier partícula del enjambre.

5. Se modifica la posición de la partícula, sumando la posición actual de la partícula con la nueva velocidad obtenida, utilizando la ecuación 2:

$$P(t+1) = P(t) + V(t+1) \quad \text{(Ecuación 2)}$$

6. Este proceso se repite hasta alcanzar el criterio de parada establecido. Un ejemplo de criterio de parada puede ser el haber alcanzado el número de iteraciones establecido al inicio.

La estructura comentada se encuentra a continuación en forma de diagrama.

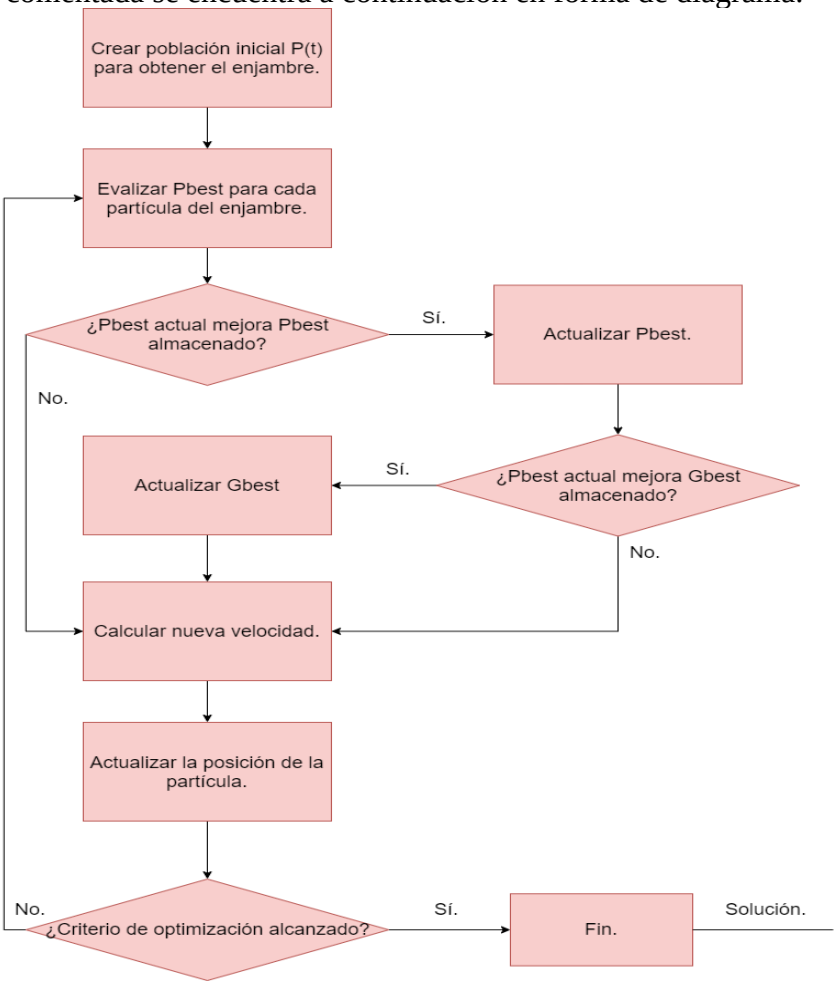

Figura 22. Estructura de un algoritmo PSO.

5.11. Algoritmo PSO KASIA.

"Knowledge Acquisition with a Swarm-Intelligence Approach" (KASIA) es un algoritmo adaptado para entrenar y generar soluciones en problemas, buscando una implementación más sencilla y de menor coste computacional que el algoritmo de aprendizaje Pittsburgh [8]. Este algoritmo es muy cercano a la idea que se tiene sobre el comportamiento de individuos dentro de un grupo o enjambre, donde el comportamiento de cada individuo o solución no es sólo dependiente de uno mismo, también del comportamiento del resto del enjambre.

Los individuos son BRs, de modo que la población está formada por bases de reglas que se evalúan de forma interna en el sistema de forma conjunta observando y almacenando qué individuo ha estado más próximo a la solución, junto a la mejor aproximación de cada miembro a esa solución.

Para realizar la función de sistema de aprendizaje, KASIA requiere de 3 elementos fundamentales.

- La población o enjambre.
- El sistema de evaluación.
- La actualización y mejora del enjambre.

5.11.1. La población.

El proceso de aprendizaje trabaja sobre una población de BR donde cada BR es una posible solución al problema. Esta población generalmente estará generada aleatoriamente, pero pueden existir casos donde exista conocimiento previo disponible para crearla.

Cada individuo debe, a su vez, tener asociado una Velocidad para que este pueda mejorar durante la ejecución del algoritmo.

5.11.2. El sistema de evaluación.

El sistema de evaluación de este algoritmo es algo diferente, pues se realiza en base a la mejor posición global Gbest y la mejor posición de un individuo, Pbest. Por ello, se dice que cada individuo es evaluado de forma conjunta, pues siempre debe tener en cuenta los resultados de otro.

5.11.3. Actualización y mejora del enjambre.

La mejora del enjambre se realiza actualizando la velocidad de cada individuo mediante la fórmula vista anteriormente y añadiendo dicha velocidad al estado o posición en el que

se encuentra el individuo. Tras la mejora, se debe evaluar de nuevo a los individuos para conocer mejoras tanto propias como globales.

5.12. Interpretabilidad de una base de reglas obtenida.

Tras la obtención de la BR por parte de los métodos mencionados en los apartados anteriores, surgen diversas cuestiones relacionadas sobre la interpretabilidad y sencillez de esta por parte de los sistemas o si el resultado obtenido es optimizable.

Para analizar la interpretabilidad, debemos tener en cuenta dos criterios:

1. La complejidad, relacionada con el número de reglas, las etiquetas por regla, el número de variables de entrada y salida, etc.
2. La semántica, relacionada con conjuntos de pertenencia difusos.

A su vez, estos dos criterios llevan a que la interpretabilidad del sistema se basa principalmente en dos variables:

1. El número de reglas. Cuanto más complejas y numerosas sean las reglas, menos interpretable será la base de reglas. Además, a mayor número de reglas más pueden activarse de forma simultánea en el proceso de inferencia.
2. El número de variables. A mayor número de variables, más reglas deberán existir, por lo que ambos niveles guardan relación. Por tal relación, la interpretabilidad es alta si ambos niveles tienen pocos elementos, pero se vuelve complejo (interpretabilidad baja) si cualquiera de los dos niveles obtiene un gran número de elementos.

El análisis previo lleva a la conclusión de que determinar de forma lingüística el índice de interpretabilidad puede ser sencillo (interpretabilidad baja si algún nivel es complejo), pero cuantificar dicho índice de interpretabilidad no es una tarea sencilla. Sin embargo, existe un Sistema Difuso Jerárquico (SDJ) compuesto por diferentes elementos para la cuantificación del índice de interpretabilidad.

5.12.1. Sistema Difuso Jerárquico.

Este sistema se basa en la Metodología Jerárquica para el análisis de interpretabilidad de la herramienta Guaje, un entorno Java para el diseño de sistemas difusos interpretables y adecuados combinando distintas herramientas de código abierto.

La estructura del SDJ se muestra en la figura 23.

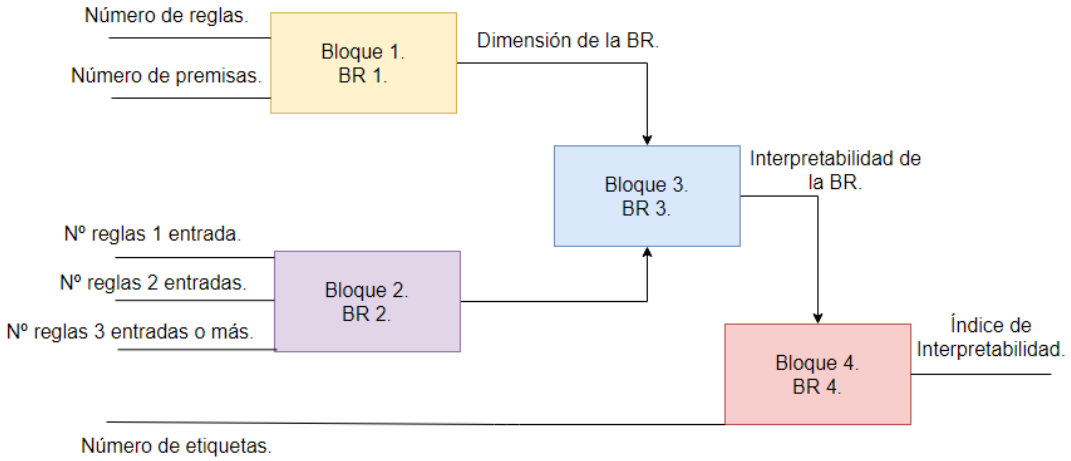

Figura 23. Estructura del Sistema Difuso Jerárquico.

5.12.2. BR1 - Dimensión de la base de reglas.

Este bloque considera el número de reglas y el número de premisas para estimar la dimensión de la base de reglas haciendo uso de reglas difusas. Las reglas difusas que se deben utilizar para la implementación de esta parte del SDJ se muestran de forma gráfica en la figura 24, donde los cuadros de color azul hacen referencia a los conjuntos difusos del número de premisas, los naranjas representan el número de reglas y el resto muestran la dimensión de la base de reglas.

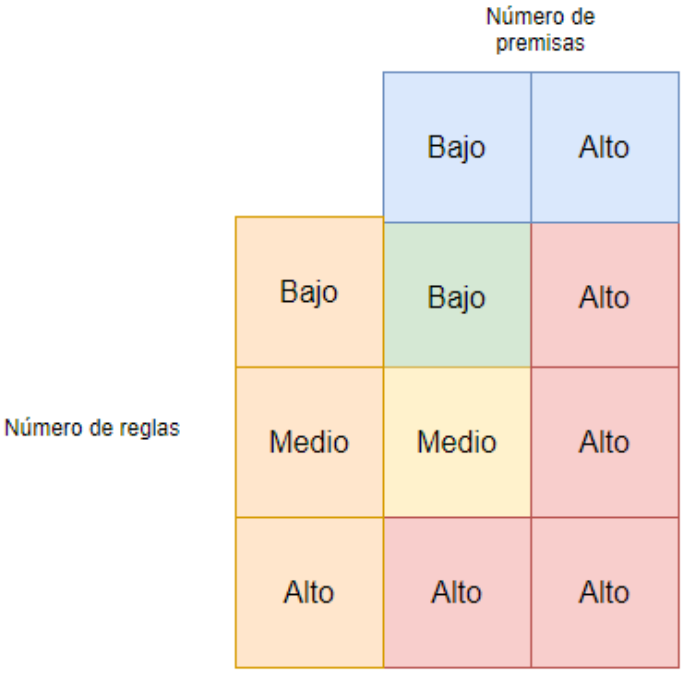

Figura 24. Reglas de BR1.

5.12.3. BR2 - Complejidad de la base de reglas.

Este bloque considera el número de variables de entrada utilizado por cada regla, clasificando las reglas existentes en 3 grupos y aplicando posteriormente un conjunto de reglas difusas para estimar la complejidad.

Los 3 grupos utilizados son:

- Número de reglas compuestas por una variable de entrada.
- Número de reglas compuestas por dos variables de entrada.
- Número de reglas compuestas por tres variables de entrada o más.

La figura 25 muestra las reglas para el bloque BR2. En la parte izquierda el número de reglas con 3 variables o más se establece como bajo, mientras que en la parte derecha es alto. Por esta razón, en la parte derecha la complejidad del sistema es mayormente alta.

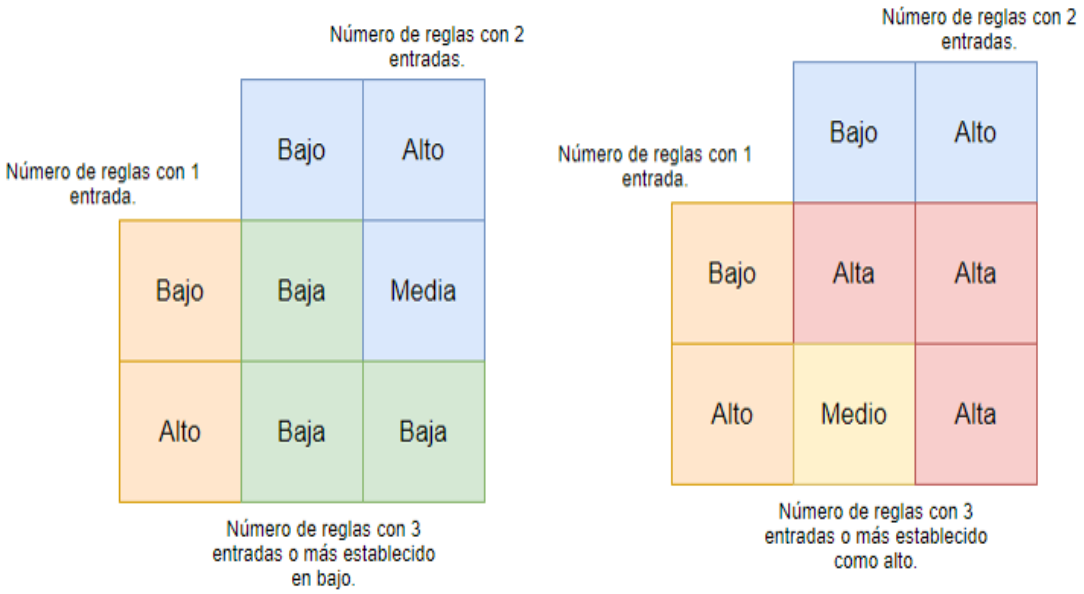

Figura 25. Reglas de BR2.

5.12.4. BR3 - Interpretabilidad de la Base de Reglas.

Este bloque recibe como entradas las salidas de BR1 y BR2 y estima la interpretabilidad aplicando su propio conjunto de reglas difusas.

La figura 26 muestra las reglas para este bloque, colocando de color azul los conjuntos de pertenencia referidos a la complejidad, en naranja los conjuntos referidos a la dimensión y el resto de los cuadros representan el conjunto borroso tomado por el consecuente, la interpretabilidad de la BR.

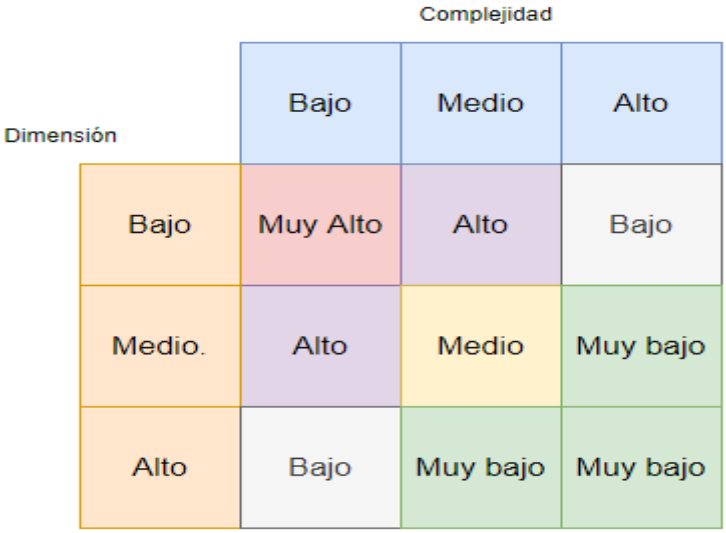

Figura 26. Reglas de BR3.

5.12.5. BR-4. Índice de interpretabilidad.

Este bloque recibe como entrada el número de etiquetas existentes en la BR junto a la salida del bloque BR3 (interpretabilidad de la BR) para obtener finalmente el índice de interpretabilidad.

La figura 27 muestra las reglas a utilizar para este bloque de una forma más visual. En este caso, los conjuntos de interpretabilidad se muestran en naranja y los referentes a el número de etiquetas en azul en la parte superior.

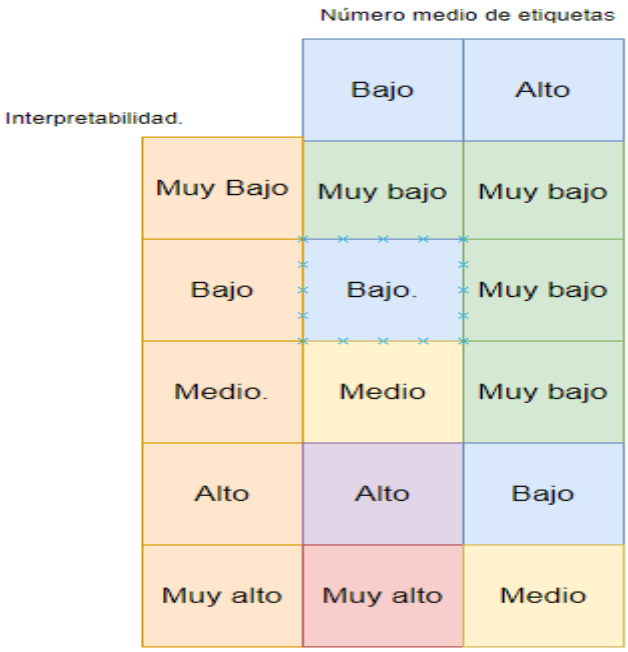

Figura 27. Reglas de BR4.

6. Métodos.

Se procede a describir de forma ordenada y esquemática el procedimiento para poder realizar de forma real los conceptos visto en el apartado anterior.

6.1. Descripción de las aplicaciones desarrolladas.

En este apartado se describen las aplicaciones desarrolladas que implementan los algoritmos y estrategias descritas en el apartado anterior. Como lenguaje de desarrollo se ha utilizado Python, ya comentado y el despliegue final se realiza en un clúster de contenedores con Docker Swarm.

6.1.1. Sistema de Inferencia Difusa.

El Sistema de Inferencia Difusa (SID) de este simulador se realiza en un archivo con formato "json" para facilitar su comprensión. Su implementación podría resumirse en los siguientes pasos:

1. Se define una variable principal, que contendrá el resto de los parámetros.
2. Se comienzan a definir las variables de "fuzzificación" y "defuzzificación" del sistema. Todas las variables y su valor deben ir entre comillas ("") para su interpretación por parte del sistema.
3. Se definen las variables de entrada al sistema junto la definición de los conjuntos de pertenencia borrosos de cada una dentro del parámetro "input".
    a. Dentro de "input", creamos otra variable llamada "name" que contendrá todos los nombres de las variables de entrada al sistema.
    b. Al finalizar "name", continuamos definiendo la variable "range", que compondrá el rango total en el que actúa una variable. Cada rango se debe definir en el mismo orden utilizado anteriormente en "name".
    c. Se definen los conjuntos de pertenencia borrosos junto a su etiqueta lingüística, la función matemática que los representa y el rango de cada conjunto dentro del grupo "membership fuzzy" (pertenencia difusa, MF).
4. Finalmente, se debe realizar el mismo procedimiento explicado en el paso 3 para la variable de salida.

6.1.2. Descripción de la base de datos.

La base de datos de la aplicación se ha realizado en lenguaje "SQL" para su manipulación y acceso a datos.

La base de datos, que en este caso se ha denominado “workflow”, contiene 2 tablas, llamadas “results” y “simulations”:

- La tabla “results” almacenará los resultados de cada iteración durante la simulación de los métodos de aprendizaje que se utilicen, como “Pittsburgh” y “KASIA”. Se compone de las variables:
    - Id: Identificador propio de los resultados, consistente en un número entero que varía si la iteración del método de aprendizaje también cambia.
    - Simulation id: Identificador de la simulación a la que pertenecen dichos resultados. Debe de ser un número entero.
    - Worker: valor alfanumérico que indica la máquina que ejecuta el proceso.
    - Iter: Iteración en la que se encontraba el proceso cuando almacenó los datos. Consiste en un número entero.
    - Results: Resultados obtenidos en la iteración del proceso. Generalmente recibirá una cadena de caracteres con los resultados.

- La tabla “simulations” almacenará parámetros referentes al tipo simulación que se esté realizando:
    - Id: identificador de la simulación, el cual es un número entero.
    - Input file: archivo DAX utilizado como entrada al simulador.
    - Scheduler: almacena el tipo de disciplina utilizada para administrar el tiempo entre procesos.
    - Type: define el tipo de simulación, ya sea una validación realizada con un sistema de reglas obtenidas anteriormente o un entrenamiento para obtener dichas reglas.
    - Learning: indica el algoritmo de aprendizaje utilizado en la simulación.
    - Goal: almacena el resultado objetivo al que pretende llegar el algoritmo.
    - Name: nombre asignado a la simulación.
    - Worker: máquina que ha realizado la simulación.
    - Status: estado actual de la simulación.
    - Bestalliter: mejor resultado de la simulación.
    - Rules: conjunto de reglas generadas por la simulación.

6.1.3. Algoritmo genético Pittsburgh desarrollado.

En este capítulo se procede a describir el desarrollo del algoritmo Pittsburgh. Para ello, se realizará en primer lugar una anotación respecto a los cambios realizados en el proyecto original y seguidamente, se describirá cada una de las partes que componen el algoritmo.

6.1.3.1. Anotación inicial: Cambios introducidos respecto al proyecto original.

En el proyecto realizado anteriormente, que da origen al descrito en este documento, se intentó la realización del algoritmo Pittsburgh. Sin embargo, este presentaba diversos fallos, como la errónea representación de los individuos de la población o una mala integración del simulador "WorkflowSim-DVFS".

6.1.3.2. Cambios en la representación de individuos.

Respecto a la representación de individuos, se tenía inicialmente una estructura errónea que imposibilitaba que el simulador ofreciera datos correctos. Esta mala representación impedía cualquier aprendizaje por parte del algoritmo, pues su aprendizaje se basa en la recombinación y mutación de individuos y estos no eran seleccionados ni almacenados correctamente.

La figura 28 detalla cómo era la estructura inicial.

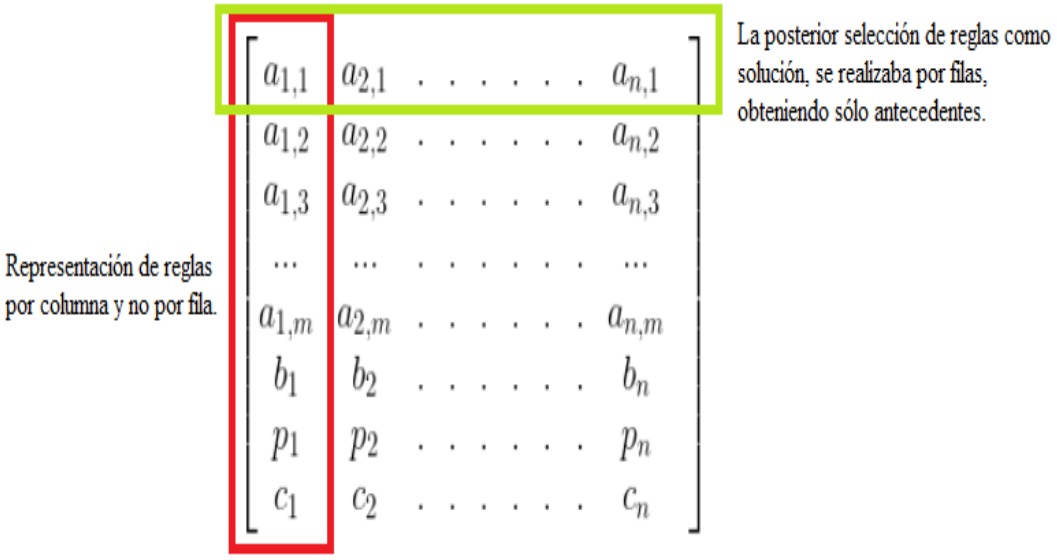

Figura 28. Detalles de estructura errónea inicial.

donde:

- La variable "n" era el número total de reglas existentes.

- La variable “m” era el número total de antecedentes existentes por cada regla.
- Las variables “a” representan los antecedentes del sistema y sus subíndices muestran la regla a la que pertenecen y qué cantidad de antecedentes hay respectivamente.
- La variable “b” representa al consecuente.
- La variable “p” indica el peso, prioridad de una regla sobre las otras.
- Por último, la variable “c” hace referencia a los conectores lógicos “and” y “or” utilizados para unir variables.

De esta forma, cada columna era una regla, cuando las reglas debían formar las filas de la respectiva matriz para que el posterior proceso de selección de filas no cogiera valores erróneos. Además, se formaban muchos individuos formados por pocas reglas en vez de pocos individuos compuestos por una gran cantidad de reglas.

La representación correcta de individuos, por su parte, debe tener la forma mostrada en la figura 29:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,m} & b_1 & p_1 & c_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,m} & b_2 & p_2 & c_2 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n,2} & a_{n,2} & a_{n,3} & \dots & a_{n,m} & b_n & p_n & c_n \end{bmatrix}$$

Figura 29. Estructura correcta del individuo.

donde:

- La variable “n” era el número total de reglas existentes.
- La variable “m” era el número total de antecedentes existentes por cada regla.
- Las variables “a” representan los antecedentes del sistema y sus subíndices muestran la regla a la que pertenecen y qué cantidad de antecedentes hay respectivamente.
- La variable “b” representa al consecuente.
- La variable “p” indica el peso, prioridad de una regla sobre las otras.

- Por último, la variable “c” hace referencia a los conectores lógicos “and” y “or” utilizados para unir variables.
- Con esta representación, cada fila sí representa una regla. Los individuos ahora están compuestos por más reglas.

6.1.3.3. Cambios en la integración del simulador “WorkflowSim-DVFS” con el algoritmo.

Inicialmente, el archivo “DAX” de entrada al simulador era seleccionado aleatoriamente en cada iteración, en vez de obedecer a la selección realizada. Este defecto ha sido corregido para funcionar con el archivo correcto desde la aplicación y además, se ha añadido una variable nueva en la base de datos para tener un registro sobre qué archivo se utilizó para realizar la simulación.

6.1.3.4. Descripción global del algoritmo.

La implementación del algoritmo en entorno Python se ha realizado en 4 partes que, aunque bastante dispares entre sí, tienen una estrecha relación. El funcionamiento general es el siguiente:

- En primer lugar, encontramos el bloque que define los distintos parámetros necesarios para formar una población aleatoria con ciertas características definidas para la ejecución del algoritmo Pittsburgh.
- A continuación, el bloque de Evaluación de individuos calificará las aptitudes de los individuos que componen nuestra población generada y guardará el sistema de reglas de los individuos como solución temporal. También se encargará de hacer una ejecución del simulador “WorkflowSim-DVFS” para la obtención de valores como el consumo de energía.
- Con los individuos evaluados, el algoritmo avanza hacia la selección de los mejores individuos de la población para finalmente aplicar los operadores genéticos, mejorando la población existente hasta la solución óptima una vez se ejecute el número de iteraciones definido en el primer bloque.

La figura 30 muestra los 4 bloques y su relación.

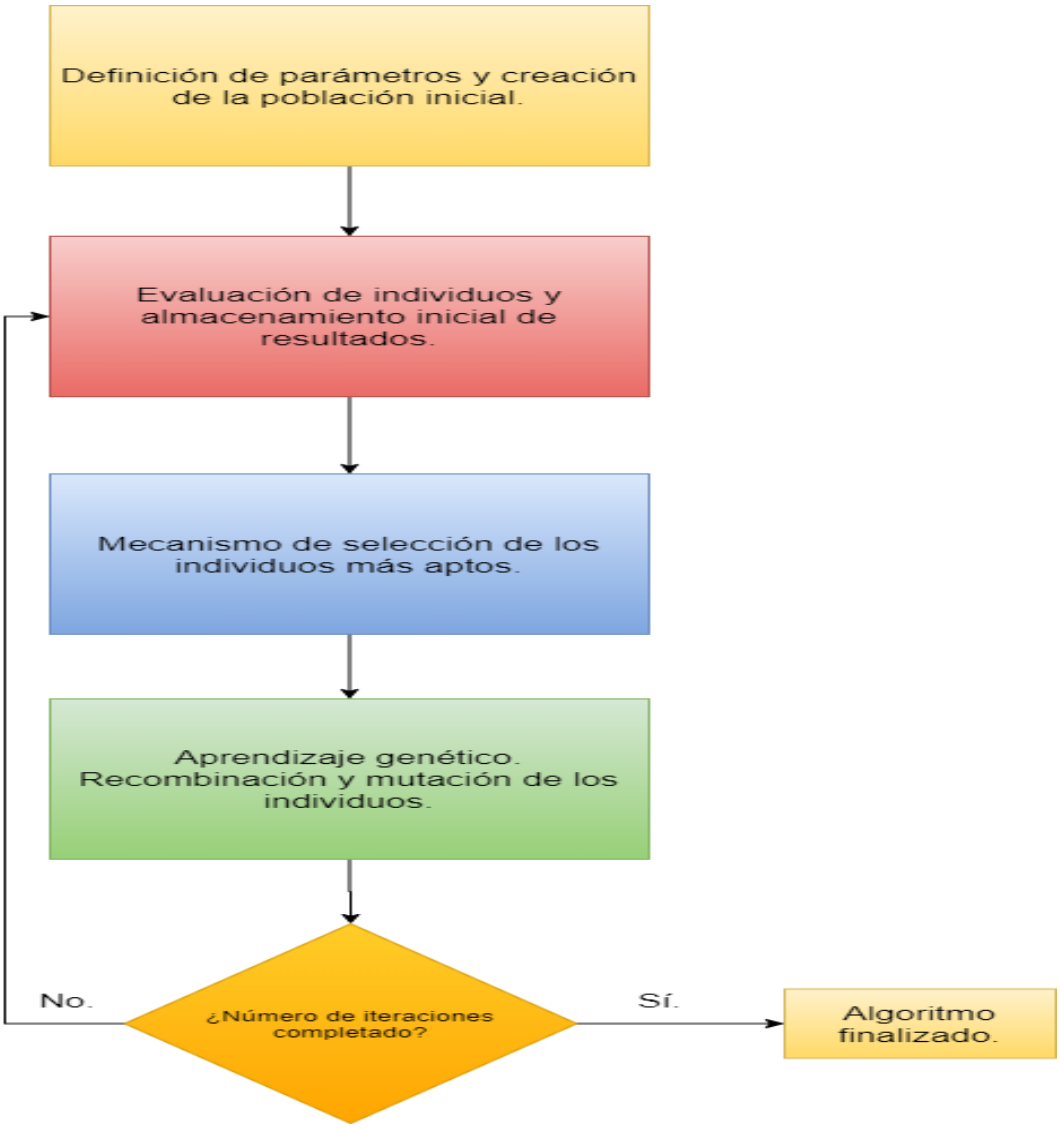

Figura 30. Estructura del algoritmo Pittsburgh desarrollado.

Los siguientes apartados muestran en detalle el funcionamiento de los 4 bloques.

6.1.3.5. Definición de parámetros y creación de la población inicial.

El bloque inicial tiene la función de definición de parámetros para los individuos. Tiene la siguiente estructura:

1. Definición de parámetros para la creación de la población inicial. Aunque esta población es aleatoria, debe tener algunas características comunes. Los parámetros que destacar son:
    a. El tamaño de la población: el número de individuos que componen nuestra muestra a evolucionar. A mayor cantidad de individuos mayor será el espacio de búsqueda cubierto por nuestro algoritmo, asegurando la consecución de una mejor solución.

b. El número de iteraciones: define el número de veces que la población será sometida a evaluación y cambios. El número de iteraciones no puede ser demasiado bajo pues puede ocasionar la finalización prematura del algoritmo.
c. Probabilidad de mutación inicial: consiste en el porcentaje de mutación que posee cada individuo al principio, ya que este porcentaje irá variando durante las diferentes iteraciones. El resto de los porcentajes de mutación tendrá como semilla esta probabilidad.
d. Porcentaje de elitismo: define qué porcentaje de individuos entrarán en la élite, considerando la élite como las mejores soluciones de la población.
e. Porcentaje de eliminación: relacionado con el porcentaje de elitismo, indica qué proporción de la población se eliminará para crear una mejor, rellenando los individuos faltantes con descendientes de la élite.
f. Número de entradas del sistema, salidas, peso y operadores que componen las reglas de cada individuo.

2. Finalmente se debe crear la población. Como se indica en la introducción teórica al algoritmo, cada individuo de la población representa una BR. Esto implica que la mejor representación de la BR sea de forma matricial.

6.1.3.6. Evaluación de individuos.

Este bloque establece la nueva probabilidad de mutación dependiente de cada iteración, pues es el bloque en el que comienza el bucle. Las reglas, comprendidas como cromosomas de un individuo, se guardarán de forma local, pero serán reescritas en las siguientes iteraciones.

En cuanto al simulador, este será ejecutado en base a una configuración, proporcionando distintos valores durante su funcionamiento, tantas veces como individuos haya existan.

La figura 31 muestra el conjunto de pasos de la evaluación de individuos.

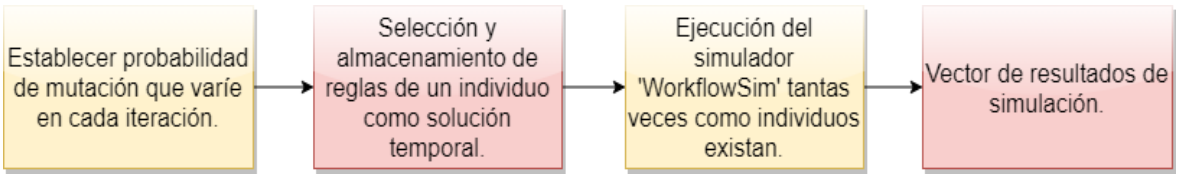

Figura 31. Evaluación de individuos para el algoritmo Pittsburgh desarrollado.

6.1.3.7. Mecanismo de selección.

Este bloque hará uso del vector de resultados obtenido anteriormente y procederá a buscar los valores mínimos para su ordenación en un vector auxiliar, guardando a su vez la posición en la que se encontraban en el de resultados. En base a este este razonamiento, tras ordenar un vector de forma creciente, el valor mínimo deberá encontrarse en la primera posición. Este valor es almacenado y comparado en cada iteración, sobrescribiendo el valor establecido como mínimo global en caso de mejoría y almacenando un nuevo bloque de reglas, ya que los primeros valores serán los mejores individuos encontrados en el proceso. La figura 32 muestra el proceso seguido.

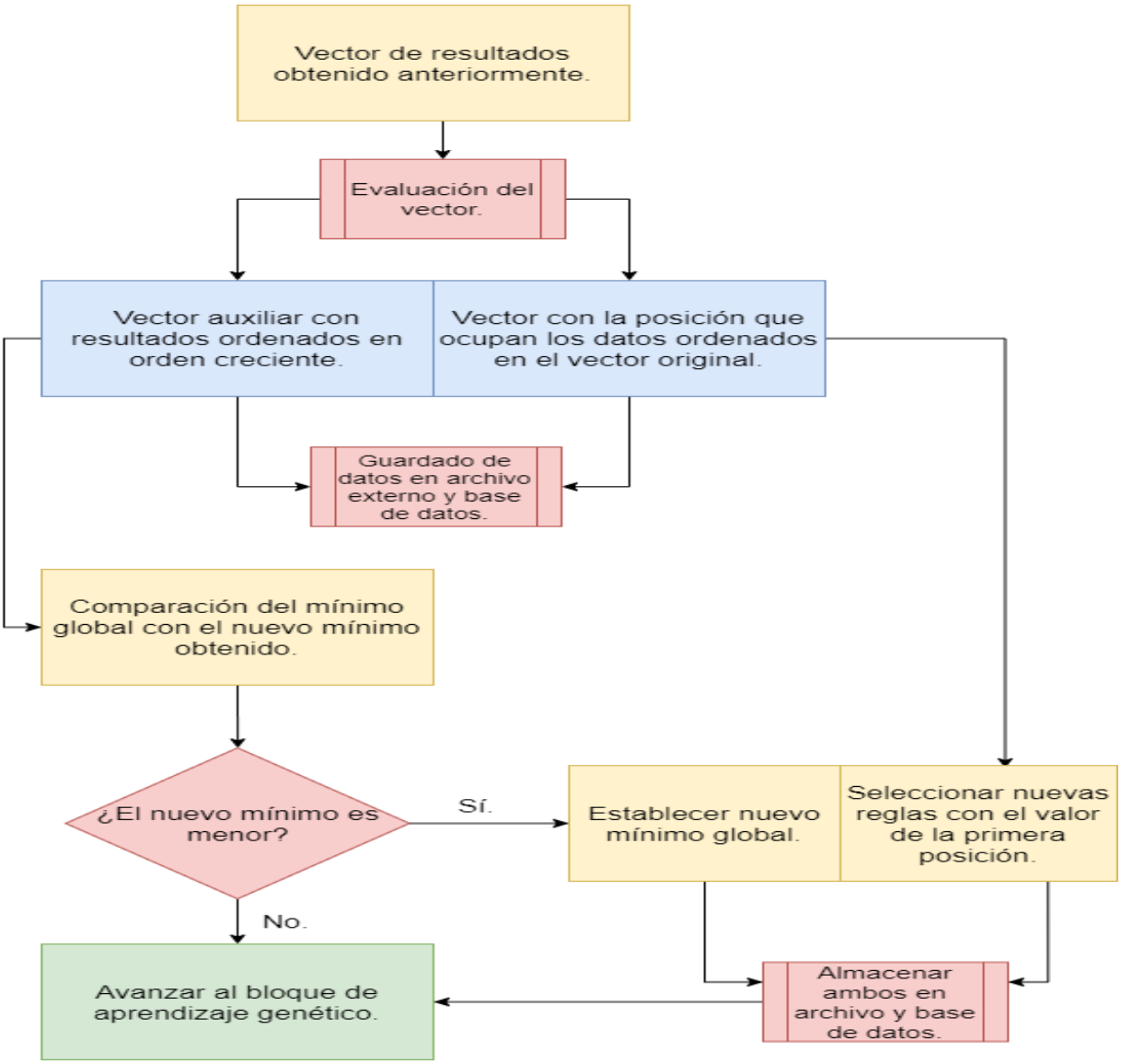

Figura 32. Mecanismo de selección de Pittsburgh desarrollado.

6.1.3.8. Aprendizaje genético.

Este bloque es el encargado de evolucionar la población que exista en ese determinado instante. Para ello hará uso de los mejores individuos, determinados por los valores almacenados en el vector de resultados del simulador junto al porcentaje de elitismo definido en el primer bloque.

Para la aplicación del operador de recombinación a los mejores individuos se sigue el siguiente procedimiento:

- Generar puntos de corte aleatorios. Estos puntos de corte deben estar redondeados al número entero más cercano.
- Intercambiar partes de filas o reglas entre los mejores individuos gracias a los puntos de corte, creando así descendientes. La definición matricial de individuos ayuda a este intercambio.

En cuanto a operadores genéticos, ya solo queda por aplicar la mutación de individuos. Dicha mutación se realiza de la siguiente forma:

- Generar un número aleatorio, que representará un intento aleatorio de mutación.
- Si dicho intento tiene un valor menor que la probabilidad de mutación existente en la iteración, se generará un segundo número aleatorio comprendido entre el número de columnas existente en cada matriz.
- Se variará el valor existente en la posición de la matriz. Dicha posición está determinada por el individuo procesado (corresponde con la matriz en sí), la regla que se estuviera iterando en ese momento (la fila de la matriz) y en último lugar, el valor obtenido en el paso anterior como columna.

Finalmente, se crea una nueva población con los descendientes obtenidos. Ahora entran en juego parámetros como:

- El porcentaje de élite que fue definido en el bloque 1.
- El porcentaje de eliminación, para saber a cuántos individuos eliminar.
- El número de iteración actual, pues en caso de haberlo alcanzado se ha finalizado el algoritmo.

La figura 33 muestra los pasos seguidos para la realización del proceso.

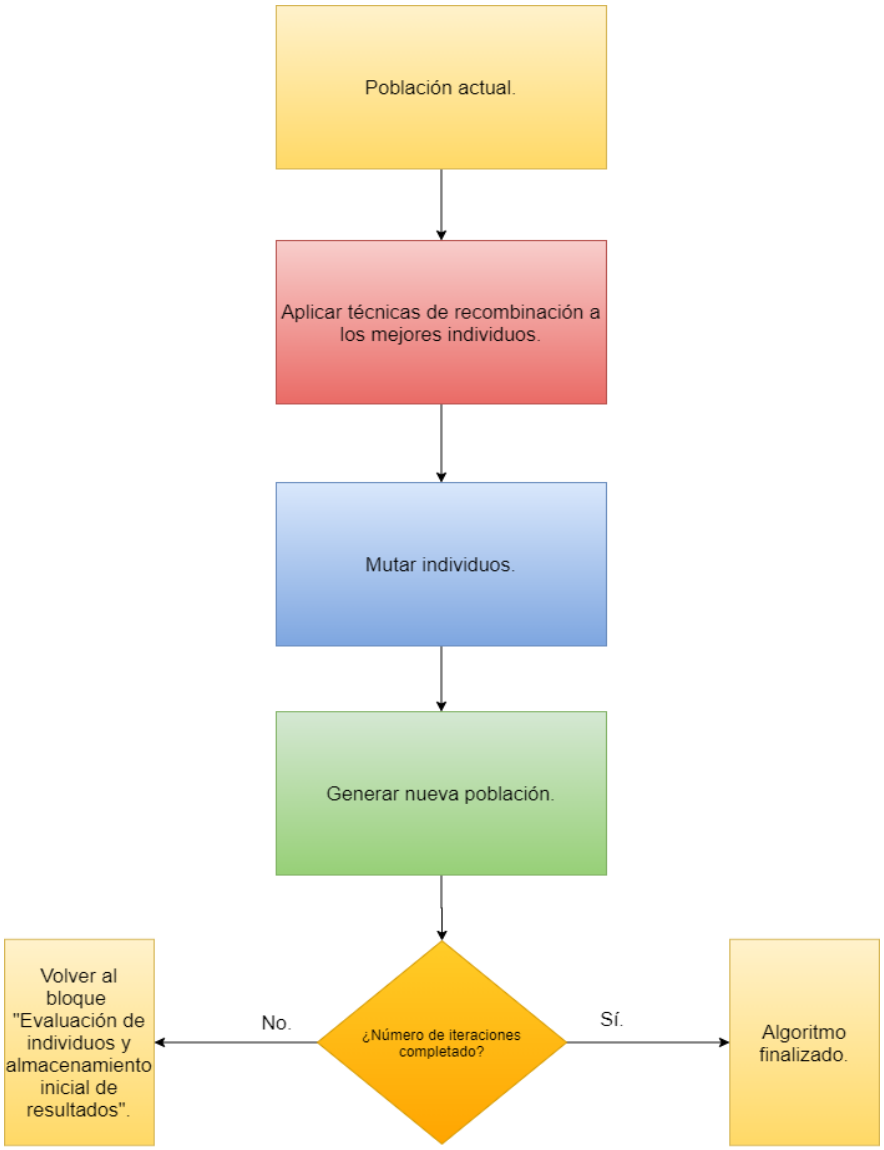

Figura 33. Aprendizaje genético del algoritmo Pittsburgh desarrollado.

6.1.4. KASIA.

El algoritmo KASIA ha sido totalmente implementado en este proyecto sin tener referencia a trabajos previos, a diferencia del algoritmo Pittsburgh.

La implementación del algoritmo en entorno Python se ha realizado en 3 bloques:

- El primer bloque hace referencia a la creación de los individuos y los parámetros de estos.
- El segundo bloque se encarga de evaluar a los individuos tras su creación.
- El tercer bloque tiene como función la actualización y mejora de individuos para establecer el proceso de aprendizaje.

Los 3 bloques, que serán definidos a lo largo de los siguientes subapartados, se muestran de forma esquemática en la figura 34.

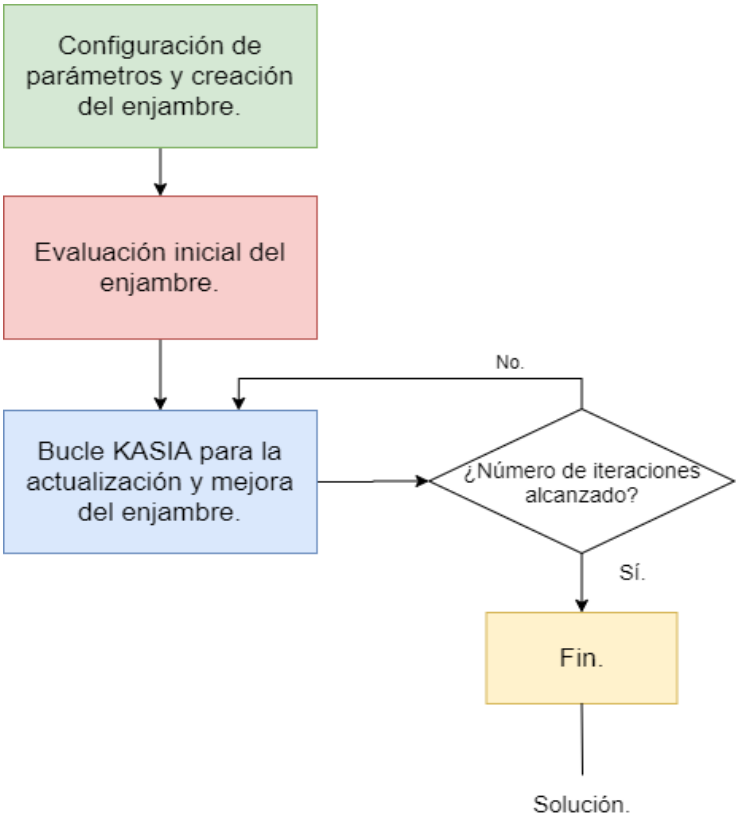

Figura 34. Estructura general del algoritmo KASIA desarrollado.

6.1.4.1. Configuración de parámetros y creación del enjambre.

En este apartado se deben definir los parámetros para la creación de individuos y con unas determinadas características. La figura 35 define de forma gráfica y general las diferentes etapas de elaboración de este apartado.

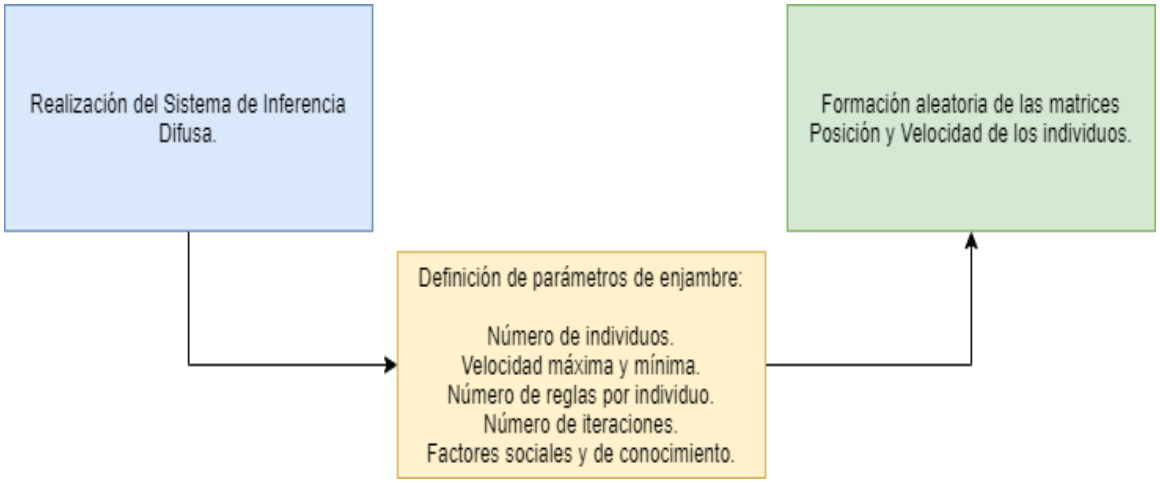

Figura 35. Configuración de parámetros y creación del enjambre.

Se define, en primer lugar, la estructura seguida para representar los elementos de Posición y Velocidad de cada individuo:

La representación correcta del elemento Posición, por su parte, debe tener la forma representada en la figura 36.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,m} & b_1 & p_1 & c_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,m} & b_2 & p_2 & c_2 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n,2} & a_{n,2} & a_{n,3} & \dots & a_{n,m} & b_n & p_n & c_n \end{bmatrix}$$

Figura 36. Representación de individuos en el algoritmo KASIA.

donde:

- La variable “n” es el número total de reglas existentes.
- La variable “m” es el número total de antecedentes existentes por cada regla.
- Las variables “a” representan los antecedentes del sistema y sus subíndices muestran la regla a la que pertenecen y qué cantidad de antecedentes hay respectivamente.
- La variable “b” representa al consecuente.
- La variable “p” indica el peso, prioridad de una regla sobre las otras.
- Por último, la variable “c” hace referencia a los conectores lógicos “and” y “or” utilizados para unir variables.

La matriz de velocidad debe de tener la misma dimensión que la matriz Posición para que ambas puedan realizar operaciones entre sí.

Respecto a los parámetros más importantes para definir dichas matrices son:

- Número de individuos: total de individuos que compondrán el enjambre.
- Número de enjambres: total de enjambres existentes en la población.
- Número de iteraciones: número de veces que se repite el proceso, para la mejora y actualización del enjambre.
- Velocidad máxima: valor máximo permitido en la matriz de velocidad.
- Velocidad mínima: valor mínimo permitido en la matriz de velocidad.
- Factor social y factor de conocimiento del enjambre: estos valores se definen como un número real y forman parte de la actualización de la matriz de velocidad.

6.1.4.2. Evaluación inicial del enjambre.

La sucesión de acciones requeridas para poder evaluar el enjambre por primera vez se muestra en la figura 37.

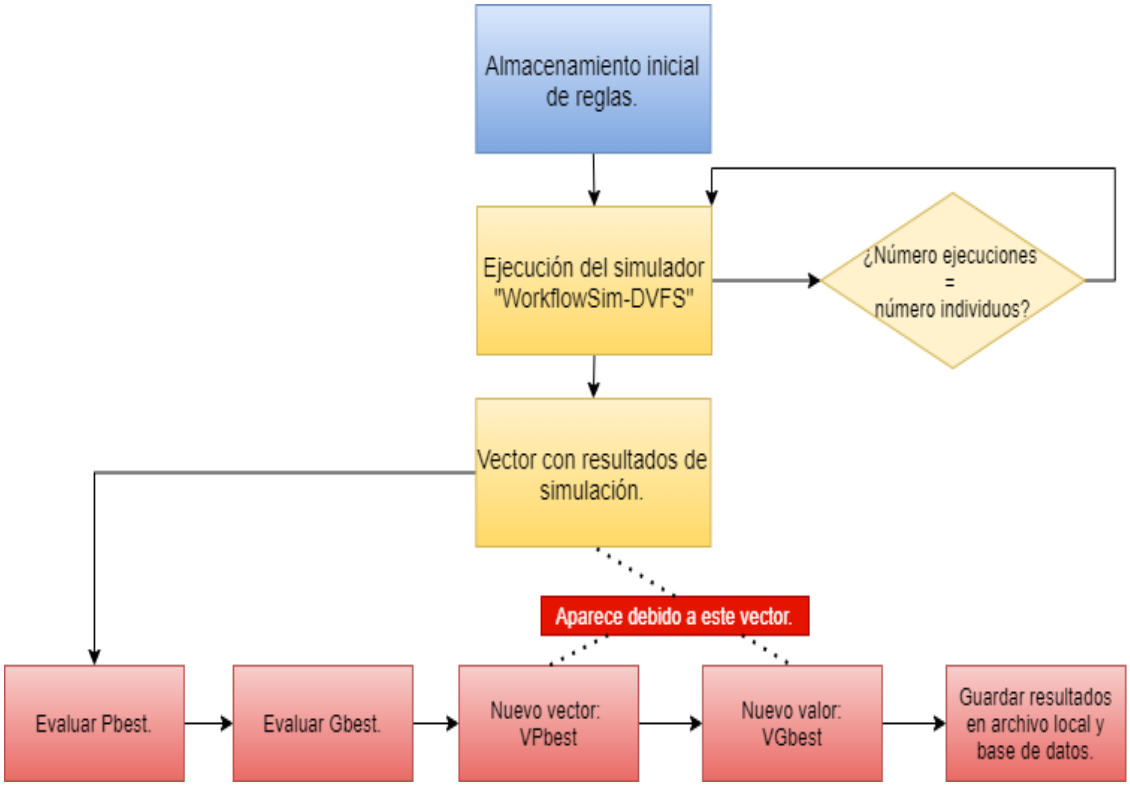

Figura 37. Esquema de evaluación inicial del enjambre.

En este bloque se debe comenzar con un almacenamiento inicial de reglas, siendo dichas reglas una matriz posición de un individuo del enjambre. De momento, no se requiere que el individuo seleccionado sea el mejor.

Seguidamente se ejecuta el simulador, tantas veces como individuos existan, para obtener un vector de resultados. En este caso, aún no se ha llegado a la actualización de individuos, de modo que la mejor configuración de cada partícula es su configuración inicial y los resultados obtenidos serán igualmente los mejores. Esta configuración inicial y vector de resultados se denominan Pbest y VPbest, respectivamente.

Para la mejor partícula del enjambre, Gbest, se procede a buscar el elemento de menor valor en VPbest, que será almacenado en una nueva variable llamada VGbest (el mejor valor global en todos los vectores VPbest), obteniendo la posición de dicho valor mínimo en el vector y situando como mejor partícula global la que se encuentre en esa posición dentro del enjambre.

Finalmente, los resultados obtenidos se deben almacenar para guardar el registro de resultados en archivos locales y en la base de datos. Estos resultados junto a los vectores y matrices obtenidos se utilizarán en el bloque final del algoritmo.

6.1.4.3. Bucle KASIA para la mejora y actualización del enjambre.

Esta parte del algoritmo KASIA tiene como principal objetivo la mejora de cada individuo del enjambre actualizando la matriz Velocidad de cada individuo y añadiéndola a la matriz Posición de su respectivo individuo. Para la actualización es necesario utilizar la fórmula vista en el apartado referido a los algoritmos PSO. La fórmula junto a algunas consideraciones se muestra a continuación:

$$V(t+1) = W * V(t) + D1 * R1 * [P best(t) - V(t)] + D2 * R2 * [Gbest(t) - V(t)]$$

- W es un valor dependiente de cada iteración, variando en cada una de estas.
- R1 y R2 deben ser matrices aleatorias con iguales dimensiones a las existentes en las matrices de Velocidad y Posición.

Tras la actualización, se debe comenzar un proceso similar al utilizado en el bloque anterior, guardando un nuevo sistema de reglas para la ejecución del simulador "WorkflowSim" tantas veces como individuos existan en el enjambre y comparar el nuevo vector obtenido con el VPbest existente. En caso de que en esta comparación existan nuevos elementos menores a los encontrados en VPbest, se actualiza la posición del vector VPbest, junto con la respectiva posición en Pbest por el nuevo individuo.

Como penúltima acción, se compara el valor mínimo en VPbest con el existente en VGbest y, en caso de que exista un valor que mejora la solución actual, se procede a la actualización de Gbest y VGbest.

Finalmente, se deben almacenar los resultados obtenidos en archivos locales y en la base de datos y comprobar si es la última iteración del algoritmo para su finalización.

El esquema general del proceso seguido en este apartado se muestra en la figura 38.

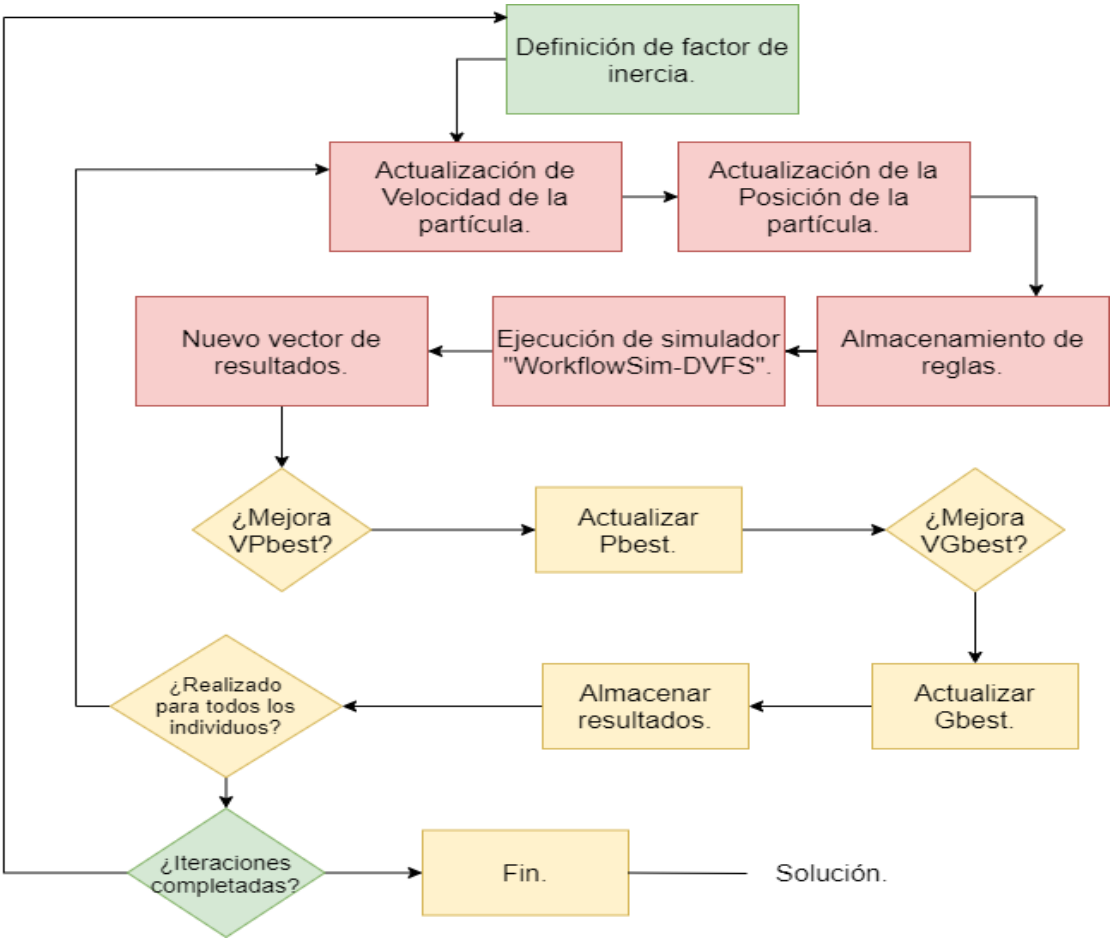

Figura 38. Bucle KASIA para la mejora y actualización de individuos.

6.1.5. Descripción del Sistema Difuso Jerárquico desarrollado.

El SDJ es el elemento necesario para obtener un índice de interpretabilidad de una base de reglas. La realización se basa en la figura vista anteriormente en el apartado referente al SDJ y por ello, en los siguientes apartados, se indicarán los pasos para la realización de este sistema en Python.

6.1.5.1. Realización del Intérprete de reglas.

Para la realización del índice de interpretabilidad de cualquier sistema de reglas de una forma automatizada, es necesario realizar una función que obtenga parámetros de la Base de Reglas dada. Los datos obtenidos por parte del intérprete serán utilizados como entrada de los distintos bloques del sistema.

El intérprete realizado cumple las siguientes capacidades:

1. Lectura de un archivo de texto, ya que este archivo será el encargado de contener las reglas en su interior en forma matricial.

2. Obtención del número de reglas existente en dicho archivo. En este caso, cada fila de la matriz del archivo de texto será una regla.
3. El cálculo del número de premisas existentes en el archivo de texto.
4. Conocer cuántas variables componen cada regla.
5. Obtención del número de etiquetas que componen cada regla.

6.1.5.2. Primer bloque del Sistema Difuso Jerárquico.

El primer bloque necesario para el desarrollo SDJ es el encargado de obtener, de forma numérica, la dimensión del sistema de reglas con los datos proporcionados por el intérprete.

Para la creación de este componente, se han realizan las siguientes acciones:

1. Se define el SID, que deberá incluir los siguientes parámetros:
    a. El número de reglas y sus respectivos conjuntos de pertenencia difusa. Esta es la primera variable antecedente del sistema.
    b. El número de premisas y sus conjuntos de pertenencia, siendo esta la segunda variable antecedente del sistema.
    c. La dimensión de la base de reglas, con los respectivos conjuntos de pertenencia, como consecuente del sistema.
2. Haciendo uso del SID creado, se deben declarar los antecedentes utilizados en el sistema, junto con sus respectivos conjuntos de pertenencia, utilizando las funciones de la librería skfuzzy y su módulo control.
3. Se declara el consecuente existente en el sistema de igual forma que los distintos antecedentes.
4. Tras la declaración de variables, se deben definir las reglas de las que hará uso el sistema difuso para realizar el proceso de inferencia.
5. Se añaden las reglas al sistema de control implementado por el módulo control de la librería.
6. Otorgando valores al sistema de control desarrollado, se simulan los resultados. Los valores de entrada serán los proporcionados por el intérprete.
7. Se obtiene la dimensión de la base de reglas como salida numérica.

La figura 39 muestra en forma de diagrama los pasos realizados.

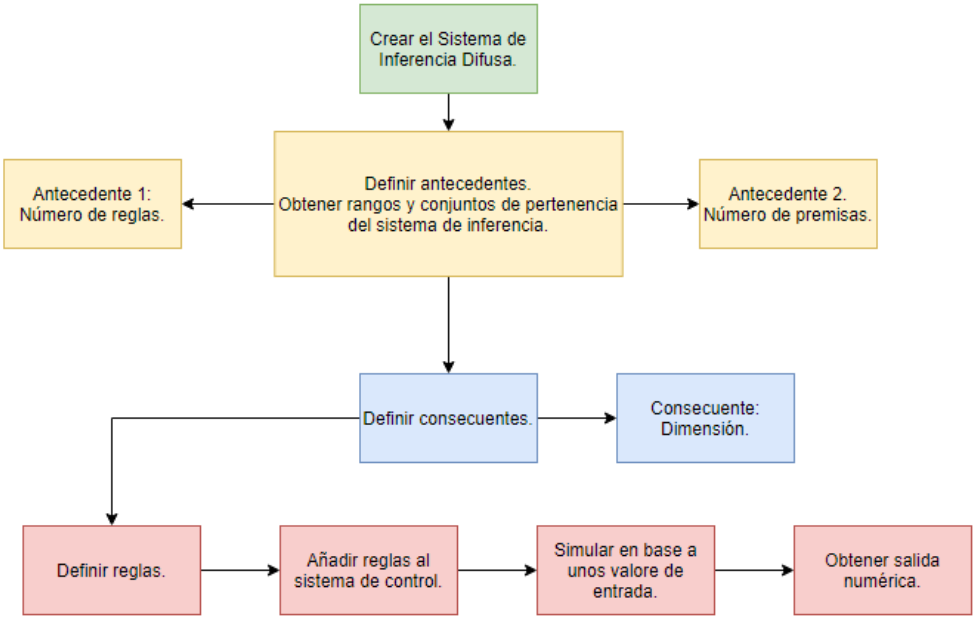

Figura 39. Estructura del primer bloque del SDJ.

6.1.5.3. Segundo bloque del Sistema Difuso Jerárquico.

Este componente tiene como objetivo determinar la complejidad de la Base de Reglas dada. Su diseño posee la siguiente estructura:

1. Se define el SID, que deberá incluir los siguientes parámetros:
    a. El número de reglas compuestas por 1 variable y sus respectivos conjuntos de pertenencia difusa. Esta es la primera variable antecedente del sistema.
    b. El número de reglas compuestas por 2 variables y sus conjuntos de pertenencia, siendo esta la segunda variable antecedente del sistema.
    c. El número de reglas compuestas por 3 variables o más y sus conjuntos de pertenencia, siendo esta la segunda variable antecedente del sistema.
    d. La complejidad de la base de reglas, con los respectivos conjuntos de pertenencia, como consecuente del sistema.
2. Haciendo uso del SID creado, se deben declarar los antecedentes utilizados en el sistema, junto con sus respectivos conjuntos de pertenencia, utilizando las funciones de la librería skfuzzy y su módulo control.
3. Se declara el consecuente existente en el sistema de igual forma que los distintos antecedentes.
4. Tras la declaración de variables, se deben definir las reglas de las que hará uso el sistema difuso para realizar el proceso de inferencia.

5. Se añaden las reglas al sistema de control implementado por el módulo control de la librería.
6. Otorgando valores al sistema de control desarrollado, se simulan los resultados. Los valores de entrada serán los proporcionados por el intérprete.
7. Se obtiene la dimensión de la base de reglas como salida numérica.

La figura 40 muestra la estructura descrita.

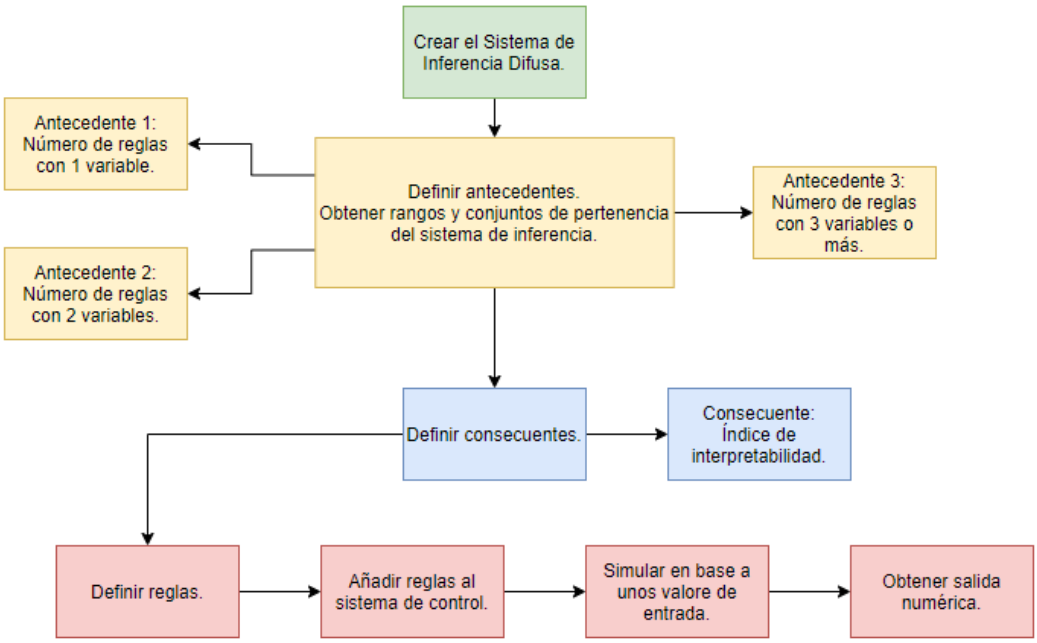

Figura 40. Estructura del segundo bloque del SDJ.

6.1.5.4. Tercer bloque del Sistema Difuso Jerárquico.

Este tercer bloque que compone el SDJ es dependiente de los 2 bloques explicados anteriormente. Recibirá como entrada numérica la salida de los 2 bloques para formar su propio resultado.

La realización de este componente se ha llevado a cabo de la siguiente forma:

1. Se define el SID, que deberá incluir los siguientes parámetros:
    a. La dimensión y sus respectivos conjuntos de pertenencia difusa. Esta es la primera variable antecedente del sistema.
    b. La complejidad y sus conjuntos de pertenencia, siendo esta la segunda variable antecedente del sistema.
    c. La interpretabilidad de la base de reglas, con los respectivos conjuntos de pertenencia, como consecuente del sistema.

2. Haciendo uso del SID creado, se deben declarar los antecedentes utilizados en el sistema, junto con sus respectivos conjuntos de pertenencia, utilizando las funciones de la librería skfuzzy y su módulo control.
3. Se declara el consecuente existente en el sistema de igual forma que los distintos antecedentes.
4. Tras la declaración de variables, se deben definir las reglas de las que hará uso el sistema difuso para realizar el proceso de inferencia.
5. Se añaden las reglas al sistema de control implementado por el módulo control de la librería.
6. Otorgando valores al sistema de control desarrollado, se simulan los resultados. Los valores de entrada para este caso serán las salidas numéricas de los bloques 1 y 2 explicados recientemente.
7. Se obtiene la dimensión de la base de reglas como salida numérica.

La figura 41 muestra en forma de diagrama los pasos realizados.

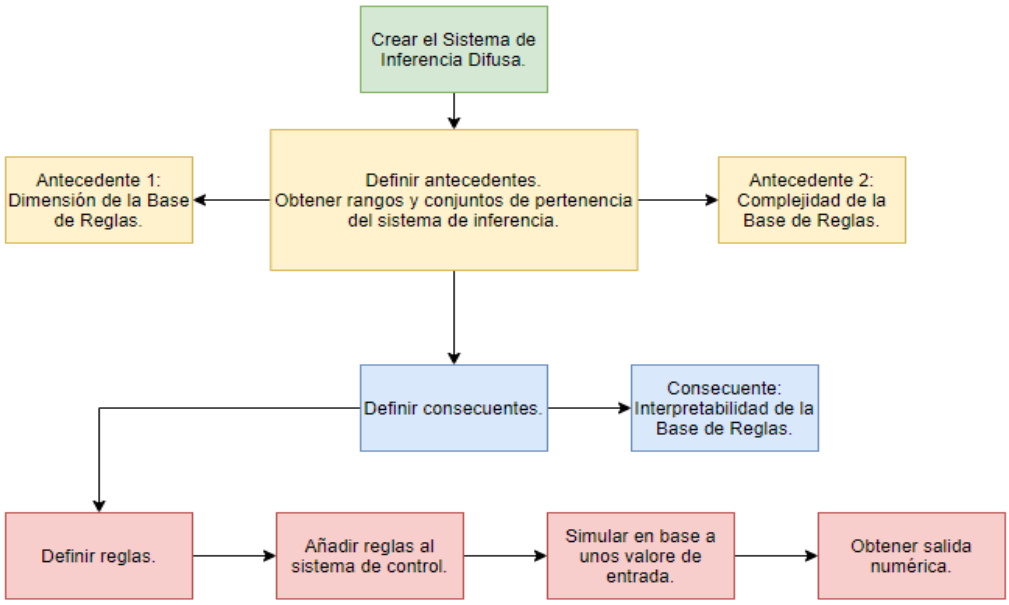

Figura 41. Estructura del tercer bloque del SDJ.

6.1.5.5. Cuarto bloque del Sistema Difuso Jerárquico.

Este apartado detalla cómo se ha realizado el bloque final del SDJ. Recibirá como entrada la salida numérica obtenida en el tercer bloque, además del número de etiquetas proporcionado por el intérprete. De esta forma, se obtiene finalmente el índice de interpretabilidad de la base de reglas.

La implementación de este componente final se ha realizado de la siguiente forma:

1. Se define el SID, que deberá incluir los siguientes parámetros:
    a. La dimensión y sus respectivos conjuntos de pertenencia difusa. Esta es la primera variable antecedente del sistema.
    b. La complejidad y sus conjuntos de pertenencia, siendo esta la segunda variable antecedente del sistema.
    c. La interpretabilidad de la base de reglas, con los respectivos conjuntos de pertenencia, como consecuente del sistema.
2. Haciendo uso del SID creado, se deben declarar los antecedentes utilizados en el sistema, junto con sus respectivos conjuntos de pertenencia, utilizando las funciones de la librería skfuzzy y su módulo control.
3. Se declara el consecuente existente en el sistema de igual forma que los distintos antecedentes.
4. Tras la declaración de variables, se deben definir las reglas de las que hará uso el sistema difuso para realizar el proceso de inferencia.
5. Se añaden las reglas al sistema de control implementado por el módulo control de la librería.
6. Otorgando valores al sistema de control desarrollado, se simulan los resultados. Los valores de entrada para este caso serán las salidas numéricas de los bloques 1 y 2 explicados recientemente.
7. Se obtiene la dimensión de la base de reglas como salida numérica.

La figura 42 muestra en forma de diagrama los pasos realizados.

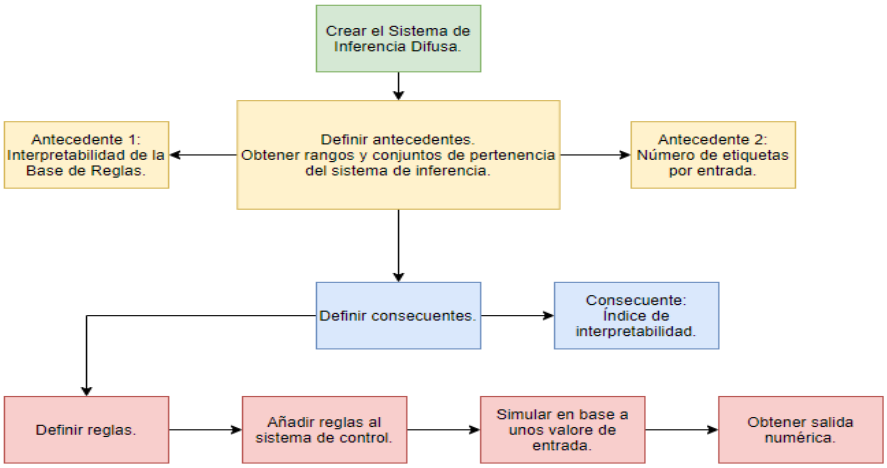

Figura 42. Estructura del segundo bloque del SDJ.

6.1.6. Aplicación web.

La aplicación realizada tiene como objetivo la comunicación de la interfaz web WFS-UI con los diferentes simuladores realizados en Python. Esta ha sido implementada con el entorno Flask y contempla multitud de casos según el recurso al que decida acceder el usuario. La aplicación se ha desarrollado con la siguiente estructura:

1. Cada vez que se inicie, la aplicación borrará la memoria caché. Esta característica evita aplicar configuraciones anteriores y, además, permite que la tabla de resultados de simulación esté actualizada.
2. Tras su inicio, en caso de acceder a la URL correspondiente desde el navegador web, se mostrará la vista principal de la interfaz WCS-UI. Desde esta vista se seguirá el procedimiento seguido en el apartado 5.3 de este documento, para ir seleccionando los parámetros de configuración de simulación.

   En general, la URL establecida por el simulador será "http://127.0.0.1:9000/", pero acepta cualquier dirección en el rango de direcciones 127.0.0.1 – 127.0.0.254.
3. Una vez se ha realizado la configuración, se seleccionará el botón "run" de la vista simulaciones. Este botón tiene asignado la ruta "http://127.0.0.1:9000/config", encargado de obtener la configuración seleccionada en la interfaz de usuario en formato JSON. La configuración pasará otro servicio alojado en "http://127.0.0.1:9000/config" y se realizará el siguiente proceso:
   a. Se procesan los datos recibidos en la petición de tipo POST, en formato JSON, y se escriben en el fichero de configuración necesario para que el simulador "WorkflowSim-DVFS" pueda ejecutarse. Seguidamente, obtendrá los valores referentes al tipo de simulación y el tipo de aprendizaje, ejecutando el tipo de simulación correspondiente.

      Para el caso concreto de la realización de una validación, las reglas pasadas como parámetros se escribirán directamente en el fichero de reglas usado por el simulador "WorkflowSim-DVFS".
   b. Las simulaciones seguirán su normal curso de funcionamiento, con su correspondiente almacenamiento de valores en la base de datos cuando sea necesario.
4. Durante la realización de las simulaciones o tras la realización de estas, se puede proceder a la tabla que muestra los resultados de simulación. Para ello basta con utilizar el botón "View All" situado en la vista de la lista de tareas, que accederá a

la ruta “http://127.0.0.1:9000/simulations” ejecutando la correspondiente función en Python:

a. Esta función se encargará de obtener todos los datos existentes en la base de datos. Los procesará y pasará de nuevo a formato json, escribiéndolos en un fichero de configuración local almacenado dentro del directorio “static” del entorno Flask.

b. El dichero con la información de la base de datos será utilizado para renderizar la vista de las tablas de simulación. Esta vista accede al fichero mediante una función AJAX de tipo “getJSON”, iterará sobre los diferentes pares de “clave-valor” contenidos y, posteriormente, rellenará la tabla con los datos correspondientes.

   Esta vista también posee una configuración que permite que se recargue automáticamente cada 10 segundos, manteniendo actualizados los datos sobre la simulación para el usuario. Esta característica no es posible sin la eliminación de la memoria caché almacenada, puesto que siempre mostrará una tabla estática hasta que se inicie una nueva sesión.

La figura 43 muestra de forma esquemática las acciones y relaciones de la aplicación web descrita.

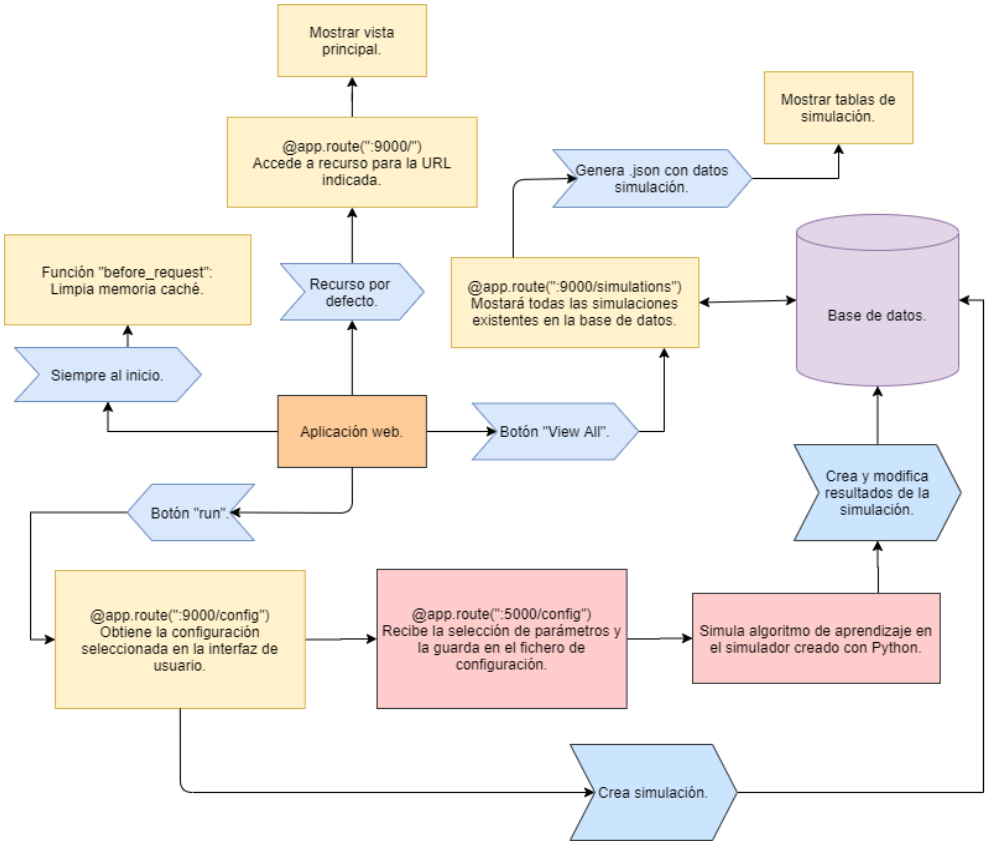

Figura 43. Arquitectura de la aplicación web desarrollada.

6.1.7. Integración de la aplicación en tecnología Docker.

Este apartado describe todos los pasos necesarios y la forma de configurar los parámetros para que la aplicación desarrollada sea ejecutada en Docker. Para ello, la aplicación se ha dividido en 3 servicios relacionados:

- El servicio de base de datos. El directorio debe tener en su interior el fichero de texto que contenga la estructura de la base de datos implementada.
- El servicio del simulador. Esta parte de la aplicación contendrá todos los archivos Python desarrollados que implementen los algoritmos de aprendizaje, los métodos de acceso a la base de datos y el proceso que maneje la petición realizada en la interfaz web. Además de estos archivos, el directorio debe de contener el simulador "WorkflowSim-DVFS" y todos sus archivos de configuración, como los ficheros DAX de entrada. Las librerías utilizadas se indican en un fichero de texto denominado "requirements.txt".
- El servicio web, que contendrá todas las vistas de la interfaz de usuario, las hojas de estilo y los archivos escritos en Python que formen el servidor y realicen acciones según el recurso solicitado. Este servicio, al estar desarrollado con la librería Flask, sigue su misma estructura y organiza los archivos en los subdirectorios "static" y "templates". Las librerías se implementan de igual forma que en el servicio simulador.

Tras la realización de este trabajo, es necesario la obtención o creación de las diferentes imágenes de los servicios.

6.1.7.1. Imágenes mysql y phpmyadmin.

Estas son las únicas imágenes obtenidas desde el directorio de descarga de imágenes docker e Implementarán la base de datos para el servicio. La imagen phpmyadmin es necesaria porque la base de datos se crea con ese entorno, usando el lenguaje sql.

6.1.7.2. Imagen del simulador.

La imagen del simulador se realiza mediante un archivo dockerfile y la construcción de este por parte de Docker. El archivo contiene las siguientes instrucciones:

- En primer lugar, se debe colocar la instrucción FROM que indica la imagen base desde la que se construirá una nueva imagen. Como el simulador está realizado en Python, se debe indicar aquí la imagen de este lenguaje en su versión utilizada.

- La instalación de OpenJDK para implementar Java y de esta forma poder ejecutar el simulador “Workflow”.
- Seguidamente, se debe copiar el archivo de dependencias, ubicado en el directorio raíz e instalar las librerías que contenga. Tras esto, se deben instalar las dependencias.
- Ahora se copian todos los archivos que componen el servicio simulador en el directorio del contenedor. Tras esta copia es necesario el cambio del directorio de trabajo al indicado para el contenedor.
- Por último, se debe ejecutar el archivo principal de la aplicación.

6.1.7.3. Imagen del servicio web.

La realización de esta imagen sigue el mismo método utilizada para crear la imagen del simulador. Las instrucciones necesarias en este dockerfile son:

- La instrucción FROM para indicar la imagen base es también una imagen Python.
- Se copia el archivo dependencias con COPY, ubicado en el directorio raíz e instalar las librerías que contenga.
- Se copian los archivos del directorio local al contenedor. Estos archivos son escritos en Python y los de los directorios templates y static de Flask.
- Cambio del directorio de trabajo al indicado para el contenedor.
- Por último, se debe ejecutar el archivo principal de la aplicación.

6.1.7.4. Manejo de la aplicación con Docker Compose.

El manejo o dirección de los servicios de la aplicación se realizan con un archivo Docker Compose y el posterior comando “docker-compose build up”. Los parámetros necesarios para la realización del archivo son:

- La red virtual de la que hará uso la aplicación.
- El servicio de la base de datos, que a su vez requiere hacer uso de las variables:
    - Imagen: imagen utilizada para crear el servicio.
    - Puertos: mapeo de puertos de la máquina local y el contenedor.
    - Redes: red que utilizará el servicio y la definición de un alias para que el resto de las aplicaciones encuentren el servicio.
    - Variables de entorno: referidas a la conexión a la base de datos, como el nombre de usuario, la contraseña, el uso de contraseña vacía o el nombre de la base de datos a usar.

- El servicio phpmyadmin.
- El servicio web, que requiere de:
    - La imagen creada para el servicio web creada anteriormente.
    - Dependencia: servicio del que depende.
    - Puertos: mapeo de puertos de la máquina local y el contenedor
    - Redes: red de la que hará uso el servicio, creada al comienzo de este apartado.
- El servicio simulador tiene los mismos parámetros que la web, cambiando el número de puerto, la imagen y añadiendo un alias.

La figura 44 muestra de forma gráfica el entorno realizado.

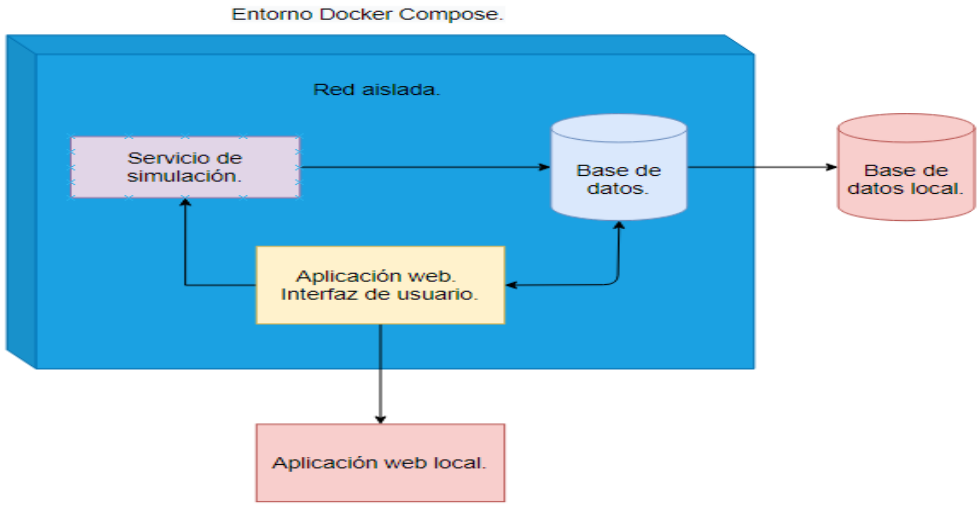

Figura 44. Entorno de Docker Compose.

6.1.7.5. Creación del clúster con Docker Swarm.

Este capítulo se enfoca en mostrar los pasos y conjunto de comandos necesarios para la creación de un clúster con Docker que incluya la aplicación realizada. La realización requiere que existan tanto administradores como trabajadores, por lo que esta será la primera tarea a realizar.

En el caso del nodo administrador, el comando utilizado para iniciar e introducir el nodo en el clúster tiene la forma:

- "docker swarm init – advertisement-addr 'ip_administrador'"
    - Este comando iniciará el enjambre de ordenadores, configurando el nodo como administrador. También realizará anuncios a los nodos que se conecten al clúster, indicando la dirección correspondiente del administrador.

En el nodo trabajador, se establece una configuración diferente. Generalmente, el propio administrador proporciona el comando para añadir los nodos, con la estructura siguiente:

- "docker swarm join –token 'identificador' 'ip_administrador':'puerto'"
    - La opción token indicara junto con el identificdor si el nodo se une como administrador o como trabajador.

Una vez que se tiene el clúster creado, es necesario proceder a la creación de las imágenes de los servicios que componen la aplicación en las 2 máquinas. El archivo yaml de Docker compose debe encontrarse en el administrador, para ser capaz de levantar toda la aplicación realizada con el siguiente comando:

- "docker stak deploy -c 'ruta_docker-compose.yml' 'nombre de stack'"
- Creará todos los servicios existentes en el fichero de forma automática.

Finalmente, se procede a la replicación de los servicios realizados. La formación de réplicas se realiza también mediante comandos, estableciendo el número de réplicas a realizar. Posteriormente, el administrador dividirá dichas réplicas del servicio entre todos los nodos existentes en el clúster, incluyéndose él mismo en la ejecución de servicios. El comando utilizado para esta tarea es:

- "docker service scale 'id_servicio' = 'número de réplicas'"

7. Resultados y discusión.

El objetivo del presente capítulo es el análisis de los diferentes resultados proporcionados por las aplicaciones realizadas. Todos los apartados seguirán la misma estructura, observando primero los resultados obtenidos e incorporando una discusión de estos.

7.1. Resultados obtenidos y discusión para el cálculo del Índice de Interpretabilidad.

Se han realizado distintas pruebas para probar el Índice de Interpretabilidad calculado por el sistema jerárquico de la figura 23, explicado en apartados anteriores. En ellas se ha variado el número de reglas, premisas, número de entradas con 1,2,3 o más entradas y el número promedio de etiquetas por variable de entrada.

Un ejemplo de índice de interpretabilidad se muestra a continuación. Se desea estimar el Índice de interpretabilidad de la siguiente BR:

```
3 1 1 1 2 4 1 1
3 2 0 0 0 1 1 1
1 0 0 0 1 4 1 1
3 2 1 2 2 1 1 1
2 1 3 2 2 5 1 1
```

El sistema BR1 tiene en cuenta el número de reglas y el número de premisas como antecedentes, estableciendo la dimensión de la BR como consecuente. Al aplicar el intérprete desarrollado a la base de reglas, se obtienen los siguientes valores para los antecedentes:

- Número de reglas: 5.
- Número de premisas: 19.

Como resultado, se obtiene una salida con valor de 0.4229 para la dimensión de la base de reglas.

El siguiente bloque del sistema, denominado BR2, considera como consecuente la complejidad de la base de reglas. Para ello, se tiene en cuenta el número de reglas con 1 variable de entrada, en número de reglas con 2 variables y el número de reglas con 3 variables o más. Aplicando el intérprete, se obtienen los siguientes valores para las variables nombradas:

- Número de reglas con 1 variable de entrada: 0.
- Número de reglas con 2 variables de entrada: 2.
- Número de reglas con 3 variables de entrada o más: 3.

Utilizando estos valores, se obtiene como resultado para la complejidad de la BR el valor 0.3230.

Ahora se definen los resultados para el tercer bloque. Este bloque es el encargado de proporcionar el valor de interpretabilidad de la BR dada. Utilizará como entradas las 2 salidas de los 2 bloques anteriores, de modo que en este caso no se obtienen valores directamente del intérprete.

El resultado obtenido es para la interpretabilidad es 0.6532.

Por último, se establecen las entradas para el bloque 4:

- Como primera entrada, se tiene el resultado del bloque 3, el valor de interpretabilidad de 0.6532.
- Como segunda entrada, se utiliza el número promedio de etiquetas por variable de entrada. Este parámetro es proporcionado por el intérprete y su valor es 3.

La salida obtenida del bloque 4, el índice de interpretabilidad que se ha buscado desde el planteamiento del SDJ este apartado, es 0.60.

A lo largo de diferentes experimentos con el sistema de interpretabilidad mostrado, se ha comprobado que el índice de interpretabilidad aumenta o decrece según varíen los parámetros obtenidos desde el intérprete, mostrando resultados coherentes. Por ello, en caso de querer obtener una buena interpretabilidad de la BR, se podría establecer que la aplicación desarrollada una buena herramienta para el análisis de la interpretabilidad, ayudando a optimizar una BR o haciendo que cualquier otro algoritmo utilice las reglas con mayor interpretabilidad.

7.2. Resultados obtenidos y discusión para el algoritmo Pittsburgh.

Como se comentó anteriormente, en este TFG se ha llevado a cabo una y modificación del código que implementa el aprendizaje de BR basadas en la aproximación de Pittsburgh. Por ello, se decidió realizar experimentos para comprobar el correcto funcionamiento del código.

Para este apartado de resultados, se han realizado una serie de experimentos para comprobar el correcto funcionamiento del código. Se han efectuado un total de 30 experimentos con la siguiente configuración:

- Tamaño de población: 20 individuos.
- Tasa de cruce: 80%.
- Probabilidad de mutación inicial: 0.1%.
- Número de iteraciones: 500.
- Número máximo de reglas por individuo: 50.
- El objetivo es la optimización de la energía consumida.
- Las variables de entrada están definidas por el simulador “Workflow” integrado, puesto que su elección no es tarea de este TFG.

El sistema cloud simulado posee 20 nodos heterogéneos con valores de potencia consumida en plena carga y en reposo como los indicados en la tabla 1. Las máquinas virtuales son homogéneas y cada una de ellas dispone de 1500 MIPS.

Nodo	1	2	3	4	5	6	7	8	9	10
Plena carga	225,00	262,89	294,35	319,82	339,73	354,49	364,50	370,14	371,80	369,83
En reposo	157,50	184,02	206,05	223,88	237,81	248,14	255,15	259,10	260,26	258,88
Nodo	11	12	13	14	15	16	17	18	19	20

Plena carga	76,94	89,90	100,66	109,37	116,18	121,23	124,65	126,58	127,15	126,47
En reposo	114,73	134,04	150,09	163,08	173,23	180,75	185,86	188,73	189,58	188,57

Tabla 1. Valores de potencia (watios) en plena carga y reposo de los nodos.

Para cada experimento, se ha seleccionado la evolución del que se ha considerado mejor individuo en cada una de las iteraciones. Con la evolución de este individuo, se ha procedido al cálculo de la media de todos ellos por iteración, obteniendo una idea del aprendizaje del algoritmo. Así mismo, en la figura 45, se puede verificar que el algoritmo desarrollado aprende y va convergiendo a valores menores de consumo de energía, aprendiendo de una manera más o menos progresiva y convergiendo finalmente en un valor.

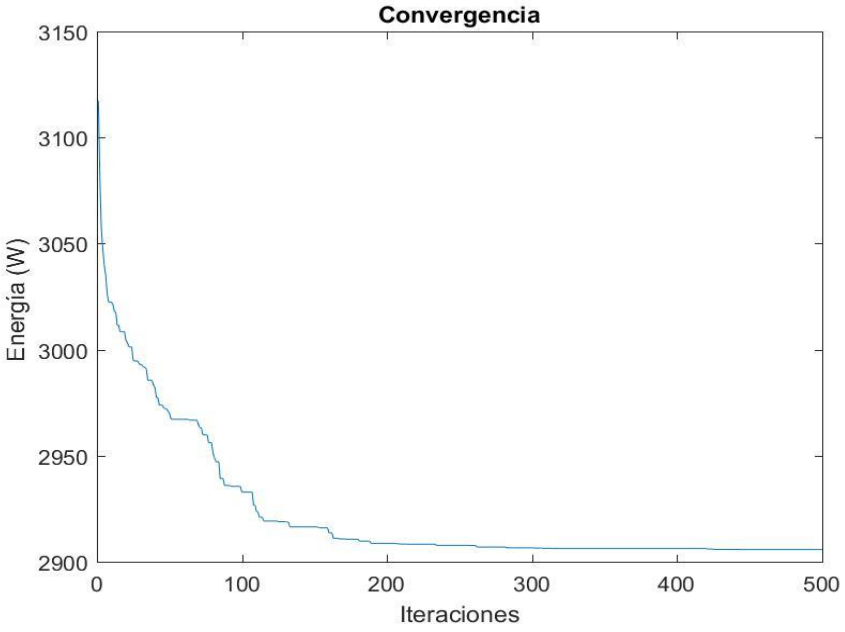

Figura 45. Curva de convergencia del algoritmo Pittsburg.

El eje X representa cada iteración, mientras que en el eje Y se muestra el valor promedio de energía obtenido por los mejores individuos de cada experimento.

El algoritmo Pittsburgh implementado, parece un funcionamiento correcto debido a los siguientes factores:

- Selecciona de forma correcta del mejor valor obtenido.
- Se producen los cruces correspondientes entre progenitores con el objeto de generar la población descendiente.
- El porcentaje de elitismo y reemplazo está bien aplicado.

- El algoritmo converge y mejora el valor de consumo de energía.

7.3. Resultados obtenidos y discusión para el algoritmo KASIA.

En este apartado se procede a la visualización de resultados proporcionados por la implementación del algoritmo KASIA. Por problemas de disponibilidad de equipos de simulación, sólo se ha podido realizar 3 experimentos, que, si bien no son suficientes para estudiar la bondad de KASIA, sí es útil para verificar que está bien implementado y realiza un proceso de aprendizaje. Un estudio más exhaustivo del algoritmo KASIA fue realizado y publicado en el artículo

La configuración utilizada para los experimentos es la siguiente:

- Tamaño de población: 20 individuos.
- Número de generaciones/iteraciones: 500.
- Número máximo de reglas por individuo: 50.
- Los coeficientes de PSO referentes a valores históricos, sociales y de conocimiento tienen el valor 2.
- El objetivo es la optimización de la energía consumida.
- Las variables de entrada están definidas por el simulador "Workflow" integrado, puesto que su elección no es tarea de este TFG.
- Número de enjambres: 1.

La implementación del algoritmo parece correcta pero optimizable debido a que:

- El algoritmo converge y mejora el valor de consumo de energía.
- La mejora automática de los individuos se lleva a cabo, por lo que el desarrollo de la matriz velocidad es correcto.
- Selecciona de forma correcta el mejor valor obtenido por un miembro del enjambre.
- El aprendizaje presenta una convergencia de aprendizaje muy escalonada.

La figura 46 muestra la curva de convergencia para el algoritmo desarrollado.

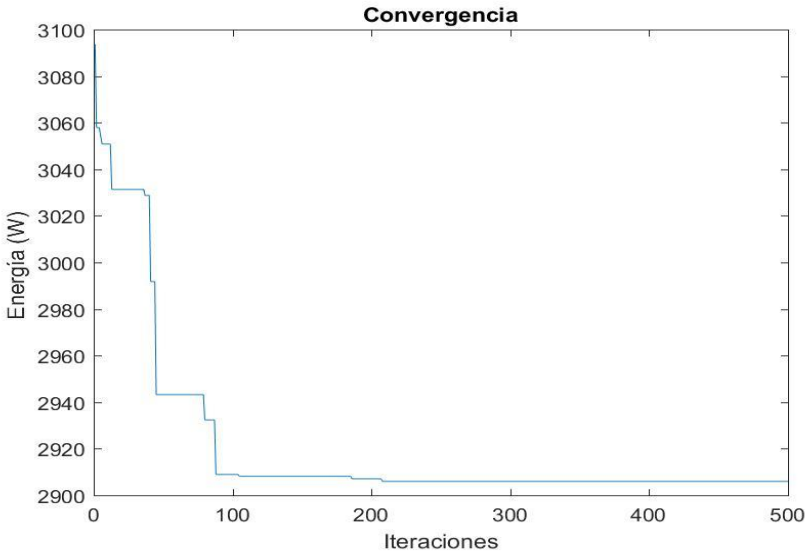

Imagen 46. Curva de convergencia del algoritmo KASIA desarrollado.

7.4. Modificaciones realizadas en el simulador y discusión sobre las modificaciones.

En la aplicación realizada anteriormente durante el TFM, los resultados se almacenaban en la base de datos utilizando servicios externos. En el proyecto actual, los datos se envían a almacenar directamente en la propia ejecución de la simulación. Para esta modificación ha sido necesario realizar diferentes funciones en Python de conexión con la base de datos. La estructura general de estas funciones junto con algún ejemplo se muestra a continuación.

- En primer lugar, se debe definir un conector con la base de datos, de forma que proporcione parámetros de acceso. La imagen 47 muestra el conector y los parámetros de ubicación de la base de datos, de nombre del usuario que accede, contraseña y nombre de la base de datos.

```
dbConnect = {
    'host': 'localhost',
    'user': 'root',
    'password': '',
    'database': 'workflow'
}
```

Imagen 47. Conector de la base de datos para Python.

- Con el conector creado, se pueden crear funciones que definan sentencias SQL y la ejecuten en la propia base de datos. La función de creación de simulaciones en la base de datos se muestra en la figura. Esta sigue un orden básico de conexión, declaración de sentencia, ejecución de sentencia y cierre de conexión, como se muestra en la figura 48.

```
def createSimulationIntoDB(input_file, scheduler, type, learning, goal, name, worker):
    conexion = mysql.connector.connect(**dbConectar, auth_plugin='mysql_native_password')

    cursor = conexion.cursor()

    sqlInsertarsim = "INSERT INTO simulations (input_file, scheduler, type,learning, goal, name,worker,bestalliter," \
                     "rules,status)values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s) "

    cursor.execute(sqlInsertarsim, (input_file, scheduler, type, learning, goal, name, worker, '', '', 'running'))

    conexion.commit()
    cursor.close()
    conexion.close()
```

Figura 48. Función de inserción en la base de datos.

Este nuevo diseño de aplicación presenta varias ventajas que lo hacen una mejor opción si se desea optimizar o implementar otro tipo de algoritmo de aprendizaje.

- Permite simplificar el código de la interfaz WCS-UI, separando la parte Python de HTML.
- Posibilita mantener la vista de resultados de simulación atualizada en cada iteración.
- Facilita la implementación del código referente al algoritmo, pues permite desarrollar este en menos archivos y con menos llamadas externas.
- Elimina redundancias producidas por la previa utilización de varios servicios de alamcenamiento externos al simulador.

7.5. Resultados obtenidos y discusión sobre el despliegue del Clúster Docker.

Este apartado se procederá a explicar cómo se ha desplegado el clúster de ordenadores con Docker swarm, las imágenes utilizadas y se observarán los resultados de este despliegue.

Para la creación del clúster se cuenta con los siguientes ordenadores:

- Un ordenador al que se ha llamado nodo 1, con un procesador de 8 núcleos y 32 GB de memoria RAM. Este ordenador de mayor capacidad será el administrador del clúster.
- Un segundo ordenador, llamado nodo 2, con un procesador de 2 núcleos y 8 GB de memoria RAM.

En primer lugar, se ejecuta el comando para la creación del clúster, de modo que el nodo 1 se coloque como administrador.

- “docker swarm init – advertisement-addr ‘ip_administrador’”

Seguidamente, se ejecuta el comando correspondiente en el nodo 2 para que se una al clúster.

Los dos nodos deben de tener las imágenes de los servicios correspondientes. Las imágenes correspondientes con SQL y phpmyadmin se encontrarán automáticamente, pero las referentes al simulador y aplicación web han sido creadas. Los archivos dockerfile mostrados a continuación siguen la estructura descrita en los apartados 6.1.7.2 y 6.1.7.3.

- En primer lugar, se observa el contenido del dockerfile del simulador en la figura 49.

```
FROM python:3.7
MAINTAINER Antonio Jimenez Sanchez <ajs00016@red.ujaen.es>


# Instalamos el juego de caracteres de idiomas
RUN apt-get update \
    && apt-get install -y \
        locales \
    && rm -rf /var/lib/apt/lists/*


# Elegimos como idioma de usuario el español
RUN sed -i -e 's/# es_ES.UTF-8 UTF-8/es_ES.UTF-8 UTF-8/' /etc/locale.gen && \
    locale-gen
ENV LANG es_ES.UTF-8
ENV LANGUAGE es_ES:es
ENV LC_ALL es_ES.UTF-8

# Install OpenJDK-11
RUN apt-get update && \
    apt-get install -y openjdk-11-jre-headless && \
    apt-get clean;


# Copiamos el archivo de dependencias y las instalamos
COPY requirements.txt /home/simulador/requiremets.txt
RUN pip install -r /home/simulador/requiremets.txt


# Se copian los archivos python del directorio local al contenedor
RUN rm -r /home/simulador/
COPY ./config /home/simulador/config/
COPY *.py /home/simulador/
COPY *.jar /home/simulador/


# Nos movemos al workdir con el propósito de encontrar los archivos auxiliares
WORKDIR /home/simulador/


# Ejecutamos la aplicación principl
CMD python /home/simulador/service.py
```

Figura 49. Contenido del archivo dockerfile para el simulador.

- Para el caso de la aplicación web desarrollada, el dockerfile se muestra en la figura 50.

```
FROM python:3.7
MAINTAINER Antonio Jimenez Sanchez <ajs00016@red.ujaen.es>


# Instalamos el juego de caracteres de idiomas
RUN apt-get update \
    && apt-get install -y \
        locales \
    && rm -rf /var/lib/apt/lists/*


# Elegimos como idioma de usuario el español .
RUN sed -i -e 's/# es_ES.UTF-8 UTF-8/es_ES.UTF-8 UTF-8/' /etc/locale.gen && \
    locale-gen
ENV LANG es_ES.UTF-8
ENV LANGUAGE es_ES:es
ENV LC_ALL es_ES.UTF-8


# Se copia el archivo dependencias y se instala.
COPY requirements.txt /home/web/requiremets.txt
RUN pip install -r /home/web/requiremets.txt


# Se copian los archivos python del directorio local al contenedor.
RUN rm -r /home/web/
COPY ./static /home/web/static/
COPY ./templates /home/web/templates/
COPY ./config /home/web/config/
COPY *.py /home/web/


# Nos movemos al workdir con el propósito de encontrar los archivos auxiliares.
WORKDIR /home/web/


# Ejecutamos la aplicación princiapl
CMD python /home/web/appweb.py
```

Figura 50. Contenido del archivo dockerfile para la aplicación web.

Una vez se tienen los archivos, se debe ejecutar el comando que construya las imágenes:

- docker build -t nombre_imagen:latest.
- Que para la imagen simulador será docker build -t simulator:latest.
- Y para la imagen de la aplicación web será docker build -t appweb:latest.

Finalizado el proceso de creación de imágenes, se debe utilizar la utilidad Docker compose para el despliegue de la aplicación, pues esta dispone de múltiples servicios y deben de estar coordinados. El archivo docker-compose debe utilizarse en el comando:

- docker stack deploy -c docker-compose.yml stack1

El archivo Docker compose creado se muestra a continuación. Su procedimiento de creación está explicado en el apartado 6.1.7.4

```yaml
version: '3'

networks:
  network1:

services:
  db:
    image: mysql:5.7
    restart: always
    ports:
      - 3306:3306
    networks:
      network1:
        aliases:
          - database
    environment:
      - MYSQL_USER=root
      - MYSQL_PASSWORD=
      - MYSQL_ROOT_PASSWORD=
      - MYSQL_DATABASE=workflow
      - MYSQL_ALLOW_EMPTY_PASSWORD=yes

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    restart: always
    depends_on:
      - database
    environment:
      - PMA_HOST=db
      - MYSQL_ROOT_PASSWORD=root
    ports:
      - 8080:80
    networks:
      - network1

  web:
    image: webapp
    restart: always
    depends_on:
      - database
    ports:
      - 80:9000
    networks:
      - network1

  simulator:
    image: simulator
    restart: always
    depends_on:
      - database
    ports:
      - 5000:5000
    networks:
      network1:
        aliases:
          - simulador
```

Figura 51. Contenido del archivo docker-compose de la aplicación.

Finalmente, se procede a la replicación de los servicios realizados. La formación de réplicas se realiza mediante un comando, estableciendo el número de réplicas que se desea realizar. El administrador dividirá dichos servicios entre todos los nodos existentes en el clúster, incluyéndose él mismo en la ejecución de servicios.

Los comandos para el contexto de simulación, formado por 10 réplicas de 3 tandas para realizar 30 simulaciones son:

- docker service scale stack1_simulator = 10
- docker service scale stack1_appweb= 1

Una vez realizado el servicio, entra en juego la discusión entre la implementación del servicio en un enjambre o en una sola máquina. Con la realización del clúster, el nodo Manager se encarga de asegurar el estado del clúster y notificar si algo no se ejecuta correctamente. Si un nodo entra en estado de mantenimiento, el nodo administrador puede crear nuevas tareas y redistribuir la carga entre el resto de los nodos disponibles [10]. Esta característica permite que el servicio siempre sea más fiable, divida las tareas de una forma bastante uniforme por el balanceo de carga y se realice de forma más rápida y óptima.

8. Conclusiones.

Analizando el contenido de la memoria, se ha tenido en cuenta la estructura común de un proyecto de investigación, con todos los elementos y conceptos utilizados, de forma que la memoria no dejase al margen ninguna parte del trabajo realizado. El documento desarrollado cuenta con una serie de apartados referentes a la LD. La teoría descrita es la utilizada a lo largo del trabajo como base para el desarrollo de la aplicación, por lo que el contenido es aprovechable para comprender por qué se usa este tipo de lógica. Además, se detallan diferentes sistemas que hacen uso de esta lógica, mostrando las ventajas de su uso, capacidades y deficiencias en la generación de reglas.

Con la exposición del problema mostrado en los sistemas basados en reglas, se introduce la CE. La teoría referente a la CE y sus diferentes ramas, como los algoritmos genéticos y evolutivos, son los conceptos que forman las bases para desarrollar todos los algoritmos de aprendizaje. Dado que estos algoritmos han sido desarrollados, el proceso de introducción a los conceptos de la CE se considera exitoso.

Respecto a los algoritmos de aprendizaje desarrollados, KASIA y Pittsburgh, estos han demostrado ser funcionales mejorando sus resultados en cada iteración y generando un nuevo sistema de reglas, por lo que la forma de implementarlos parece correcta. Además,

se han establecido diversos cambios, como la reestructuración de los individuos en el algoritmo Pittsburgh o la forma que estos almacenan información en la base de datos. Esta nueva implementación almacena los datos de una manera óptima, mejorando el tiempo de ejecución necesario y el número de interacciones entre procesos.

El intérprete de reglas desarrollado para la obtención del índice de interpretabilidad ofrece resultados fiables, acordes a la variación de parámetros de las reglas gracias a la división jerárquica. Esta cualidad lo convierte en un software útil para analizar las reglas obtenidas con los algoritmos desarrollados, que forman parte del resto de la aplicación final.

Es de gran importancia nombrar el uso del simulador "WorkflowSim-DVFS". Este simulador es el encargado de generar resultados a optimizar mediante los algoritmos implementados. Cada vez más servicios se desarrollan en la nube, aumentando la importancia de los centros de procesado de datos, por ello su integración y desarrollo es importante para mejorar la eficacia de consumo de estos. La mejora de sus resultados con los algoritmos era una de las tareas más complicadas e importantes del proyecto.

Es de destacar la mejora de la aplicación web desarrollada respecto a trabajos anteriores. El requisito principal era que la aplicación fuera integrable con una interfaz y con los algoritmos que se desarrollaran. Estos objetivos se han superado, pues la aplicación actual está integrada totalmente en un sólo lenguaje y, además, ofrece nuevas capacidades que hacen más fácil el seguimiento de los resultados y la obtención de estos desde la propia interfaz web. La facilidad de uso de la interfaz no se ha visto comprometida, pues utiliza acciones sencillas como la descarga de los resultados pulsando un enlace, con una visualización de resultados sencilla y limpia.

Finalmente, el trabajo desarrollado presenta una arquitectura flexible que permite un sistema escalable, dinámico y distribuido utilizable en un clúster para una realización de tareas de una forma más rápida. Se han podido aprovechar las utilidades del clúster Docker para la realización de una gran cantidad de simulaciones de los diferentes algoritmos implementados para observar resultados.

9. Abreviaturas.

- LD: Lógica Difusa.
- BR:Base de Reglas.
- TFG: Trabajo de Fin de Grado.

- SDJ: Sistema Difuso Jerárquico.
- KASIA: Knowledge Acquisition with a Swarm-Intelligence Approach.
- PSO: Particle Swarm Optimization.
- CE: Computación evolutiva.
- BD: Base de Datos.
- AG: Algoritmo Genético.
- BC: Base de conocimiento.
- DVFS: Dynamic Voltage Frecuency Scaling.
- SBRD: Sistema Basado en Reglas Difusas.
- WCS-UI: Web Cloud Simulator User Interface.

10. Referencias bibliográficas.

[1] Python. Lenguajes de programación, [Referenciado el 17 de noviembre de 2020] https://lenguajesdeprogramacion.net/python/

[2] Numpy. Numpy documentation, What is Numpy? [Referenciado el 17 de noviembre de 2020] https://numpy.org/doc/stable/user/whatisnumpy.html

[3] Palletsprojects. Flask. [Referenciado el 17 de noviembre de 2020] https://palletsprojects.com/p/flask/

[4] Pythonhosted. Skfuzzy 02 docs. [Referenciado el 17 de noviembre de 2020] https://pythonhosted.org/scikit-fuzzy/overview.html

[5] Pythonhosted. Module: control. [Referenciado el 17 de noviembre de 2020] https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.control.html

[6] Ecured. Pycharm. [Referenciado el 17 de noviembre de 2020] https://www.ecured.cu/Pycharm

[7] Iván Tomás Cotes-Ruiz, Rocío P. Prado, Sebastián García-Galán, José Enrique Muñoz- Expósito, Nicolás Ruiz-Reyes, Dynamic Voltage Frequency Scaling Simulatorfor Real Workflows Energy-Aware Management in Green Cloud Computing, Plos one, 2017.

[8] R. P. Prado, S. García-Galán, J. E. Muñoz Expósito and A. J. Yuste, Knowledge Acquisition in Fuzzy-Rule-Based-Systems With Particle-Swarm Optimization, IEEE TRANSACTIONS ON FUZZY SYSTEMS, VOL. 18, NO. 6, DECEMBER 2010.

[9] O. Cordón, L. Magdalena. F. Herrera, F. Hoddmann. Genetic Fuzzy Systems. Evolutionary Tuning and learning of fuzzy knowledge bases.

[10] Profile. Orquestadores Docker. [Referenciado el 17 de noviembre de 2020] https://profile.es/blog/comparativa-de-orquestadores-docker-swarm-vs-kubernetes-vs-apache-mesos/

[11] Amazon. Docker. [Referenciado el 17 de noviembre de 2020] https://aws.amazon.com/es/docker/

[12] Docker. Docker overview. [Referenciado el 17 de noviembre de 2020] https://docs.docker.com/get-started/overview/

[13] Docker. Dockerfile reference. [Referenciado el 17 de noviembre de 2020] https://docs.docker.com/engine/reference/builder/

[14] Docker. Docker build. [Referenciado el 17 de noviembre de 2020] https://docs.docker.com/engine/reference/commandline/build/

[15] Docker. Overview of Docker Compose. [Referenciado el 17 de noviembre de 2020] https://docs.docker.com/compose/

[16] Docker. Swarm mode overview. [Referenciado el 17 de noviembre de 2020] https://docs.docker.com/engine/swarm/

[17] Linux. CPUFreq Governor. [Referenciado el 17 de noviembre de 2020] https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt