Escuela Politécnica Superior de Jaén
Máster Universitario en Ingeniería Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Máster

SISTEMA PARA LA VISUALIZACIÓN E INTERPRETACIÓN DEL PROCESO DE INFERENCIA EN SISTEMAS EXPERTOS BASADOS EN MAPAS BORROSOS COGNITIVOS

Alumno: Garzón Casado, Álvaro

Tutores: Prof. D. Pablo Cano Marchal
Prof. D. Diego Manuel Martínez Gila
Dpto.: Ingeniería Electrónica y Automática

Septiembre, 2017

Universidad de Jaén

ESCUELA POLITÉCNICA SUPERIOR DE JAÉN

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA

TRABAJO FIN DE MÁSTER:

SISTEMA PARA LA VISUALIZACIÓN E INTERPRETACIÓN DEL PROCESO DE INFERENCIA EN SISTEMAS EXPERTOS BASADOS EN MAPAS BORROSOS COGNITIVOS

Realizado por:
Álvaro Garzón Casado
2017

Dirigido por:
Dr. D. Pablo Cano Marchal
Dr. D. Diego Manuel Martínez Gila

Índice general

Capítulos **Página**

1. INTRODUCCIÓN **1**

1.1. OBJETO DEL PROYECTO 1

1.2. MOTIVACIÓN 1

1.3. JUSTIFICACIÓN 2

1.4. ESTRUCTURA DEL PROYECTO 2

2. LÓGICA DIFUSA **4**

2.1. ¿Qué es la *Lógica Difusa*? 4

2.2. Antecedentes 5

2.3. Características 7

2.4. Sistemas Expertos 9

2.4.1. Dendral 12

2.4.2. Mycin 13

2.4.3. XCon 13

2.4.4. Watson 14

3. ESTADO INICIAL **16**

3.1. Caso de estudio: proceso de elaboración de aceite de oliva virgen (PEAOV) 18
3.2. Proceso de cálculo 26
3.2.1. Ejemplo práctico de cálculo 29
3.2.2. Modificación del código de cálculo 32

4. ALGORITMIA DESARROLLADA 37

4.1. Bloque de representación 40
4.1.1. Asignación de colores a nodos 40
4.1.2. Desarrollo de gráficos de barras en nodos sucesores 46
4.1.3. Modificaciones adicionales 50
4.2. Bloque de simplificación 55
4.2.1. Extracción de nodos y simplificación del sistema 55
4.2.2. Extracción completa 63
4.2.3. Parametrización de nodos 65
4.2.4. Cálculo de peso total internodal 70
4.3. Resumen de utilidades 74

5. CONCLUSIONES 75

6. TRABAJO FUTURO 77

I. Documentación del código empleado 79

I.1. Documentación de clases 80
I.2. Diagramas de llamadas 84

II. Modificaciones adicionales 88

III.Extracción de nodos 94

Bibliografía **108**

Índice de figuras

2.1. Función difusa alto-bajo. 5
2.2. Lofti Zadeh. 6
2.3. Ebrahim Mamdani. 6
2.4. Esquema de diseño y empleo de un sistema experto. 9
2.5. Superordenador Watson, IBM. 14

3.1. Representación gráfica de un sistema generado por el método *gen_graph*. . 17
3.2. Modelo de separación sólido-líquido. 24
3.3. Modelo de preparación de la pasta. 25
3.4. Subgrafo para ejemplo de cálculo de valor nítido. 29

4.1. Diagrama de flujo del algoritmo de asignación de colores: bloque principal. 42
4.2. Diagrama de flujo del algoritmo de asignación de colores: bloque secudario. 43
4.3. Ejemplo de asignación incorrecta de los colores. 45
4.4. Ejemplo de asignación correcta de los colores. 45
4.5. Gráfico de barras del nodo Grado Molienda. 47
4.6. Diagrama de flujo del algoritmo de generación de gráficos de barras. . . . 47
4.7. Diagrama de flujo del algoritmo de generación de carpetas contenedoras. 49
4.8. Ejemplo 1: simplificación del nodo *Emulsión Pasta Corregida*. 52

4.9. Ejemplo 2: simplificación del nodo *Incremento Temperatura Molino*. . . . 53

4.10. Representación gráfica de un sistema generado por el método *visualize_results_graph*. 54

4.11. Diagrama de flujo del algoritmo de extracción y representación del subgrafo. 56

4.12. Diagrama de flujo del algoritmo de recorrido de árbol. Inicio. 57

4.13. Diagrama de flujo del algoritmo de recorrido de árbol. Modo *backward*. . 57

4.14. Diagrama de flujo del algoritmo de recorrido de árbol. Modo *forward*. . . 58

4.15. Análisis *forward* de grado 2 de *Madurez*. 61

4.16. Diagrama de flujo del algoritmo de extracción completa 64

4.17. Diagrama de flujo del algoritmo de parametrización de nodos. 67

4.18. Ejemplo 1: parametrización de Estado Batido. 68

4.19. Ejemplo 2: parametrización de Nivel Defecto. 68

4.20. Ejemplo 3: parametrización de Nivel Frutado. 69

4.21. Ejemplo 4: parametrización de Velocidad Molino. 69

4.22. Ejemplo de grafo para cálculo de pesos totales. 71

4.23. Caminos posibles entre los nodos x e y. 71

I.1. Diagrama de llamadas del método *gen_graph*. 84

I.2. Diagrama de llamadas del método *visualize_results_graph*. 84

I.3. Diagrama de llamadas del método *extract_node*. 85

I.4. Diagrama de llamadas del método *extract_all*. 85

I.5. Diagrama de llamadas del método *parametric*. 86

I.6. Diagrama de llamadas del método *get_total_weight*. 87

II.1. Ejemplo 1: modelo de preparación de la pasta con paleta caliente. 89

II.2. Ejemplo 2: modelo de preparación de la pasta con paleta fria. 90

II.3. Ejemplo 3: modelo de separación líquido-sólido con paleta brillante. . . . 91

II.4. Ejemplo 4: modelo de separación líquido-sólido con paleta otoñal. 92

II.5. Ejemplo 5: modelo conjunto con asignación aleatoria. 93

III.1. Ejemplo 1: análisis *backward* de grado 1 de *Estado Batido*. 95

III.2. Ejemplo 2: análisis *backward* de grado 2 de *Estado Batido*. 96

III.3. Ejemplo 3: análisis *backward* de grado 3 de *Estado Batido*. 97

III.4. Ejemplo 4: análisis *backward* de grado 4 de *Estado Batido*. 98

III.5. Ejemplo 5: análisis *backward* de grado 5 de *Estado Batido*. 99

III.6. Ejemplo 6: análisis *backward* de grado 1 de *Estado Batido* y *Nivel Frutado*.100

III.7. Ejemplo 7: análisis *backward* de grado 1 de *Humedad Aceituna* y *Nivel Frutado*. 101

III.8. Ejemplo 8: análisis *forward* de grado 1 de *Madurez*. 102

III.9. Ejemplo 9: análisis *forward* de grado 2 de *Madurez*. 103

III.10. Ejemplo 10: análisis *forward* de grado 3 de *Madurez*. 104

III.11. Ejemplo 11: análisis *complete* de grado 1 de *Emulsion Pasta*. 105

III.12. Ejemplo 12: análisis *complete* de grado 2 de *Emulsion Pasta*. 106

III.13. Ejemplo 13: análisis *complete* de grado 3 de *Emulsion Pasta*. 107

Índice de tablas

3.1. Valores nítidos de los nodos iniciales de los subsistemas. 20

4.1. Paleta de colores: colores cálidos. 41

4.2. Paleta de colores: colores frios. 41

4.3. Paleta de colores: colores brillantes. 41

4.4. Paleta de colores: colores otoñales. 41

4.5. Comparativa de colores de la fuente del nodo. 50

4.6. Pesos totales internodales para todas las combinaciones *Nodo Inicial* - *Nodo final* del caso de estudio. 73

4.7. Resumen de las utilidades desarrolladas. 74

Capítulo 1

INTRODUCCIÓN

1.1. OBJETO DEL PROYECTO

El objetivo del presente proyecto es el desarrollo de un sistema para la visualización e interpretación del proceso de inferencia en sistemas expertos basados en Mapas Borrosos Cognitivos, así como su aplicación a un caso de estudio.

Se emplean como bases del proyecto las librerías estándar del lenguaje de programación Python y las librerías personalizadas de la tesis doctoral del Dr. Pablo Cano Marchal, *Contribution to the Modeling and Automatic Control of the Virgin Olive Oil Elaboration Process* [Mar15]. Sobre los casos de estudio desarrollados en la tesis mencionada serán sobre los que se trabaje en el presente proyecto.

1.2. MOTIVACIÓN

Este trabajo se enmarca dentro de los requerimientos oficialmente estipulados para la obtención del título de Máster Universitario en Ingeniería Industrial.

Como motivación fundamental cabe destacar el deseo propio del autor del desarrollo de un proyecto industrializable y con alta viabilidad de transferencia empresarial, así co-

mo el empleo de un lenguaje de programación distinto al impartido durante la titulación y la incursión en una materia no afín con otras estudiadas con anterioridad.

1.3. JUSTIFICACIÓN

La **interpretabilidad** es una de las características más interesantes dentro de los sistemas de inferencia borrosos. Sin embargo, es, a su misma vez, una de las características más difíciles de representar fielmente y de forma precisa.

Por lo tanto, el desarrollo e implementación de este sistema se justifica bajo la necesidad de un sistema visual, práctico y simple que permita la interpretación de la fase de inferencia en sistemas expertos, de forma que pueda omitirse el concepto de *caja negra* que siempre viene asociado a este tipo de sistemas, de manera que cualquier usuario sin formación en materias afines sea capaz de interpretar sin problema el sistema, es decir, se consiga una adecuada interpretabilidad.

De igual forma el proyecto se ve justificado debido al desarrollo de diferentes utilidades que aprovechan el potencial de las librerías anteriormente mencionadas, tanto para simplificar la comprensión del sistema experto como para la obtención de información importante referente al mismo.

1.4. ESTRUCTURA DEL PROYECTO

El presente trabajo se organiza de la siguiente forma: en el Capítulo 1 se han expuesto las secciones básicas de todo Trabajo Fin de Máster, incluyendo estas el objetivo perseguido en el trabajo, la motivación personal que lo conlleva y su justificación dentro del contexto del desarrollo actual de los sistemas expertos basados en mapas borrosos cognitivos.

El Capítulo 2 nos presenta una introducción a la lógica difusa, que permite al lector obtener una perspectiva de la materia, así como sus antecedentes, características y su

relación con los sistemas expertos. Así mismo, para una mayor comprensión de la trascendencia del empleo de sistemas expertos se presentan una serie de ejemplos históricos de los mismos.

El Capítulo 3 nos pone en antecedentes de nuestro proyecto, explicando el estado inicial tanto del sistema experto que vamos a estudiar como de la materia objeto del sistema, en este caso el **proceso de elaboración de aceite de oliva virgen (PEAOV)**. En este mismo capítulo se exponen los cambios básicos realizados en el sistema de cálculo del sistema experto.

El Capítulo 4 se centra en la explicación de la algoritmia desarrollada en el proyecto, especialmente en los algoritmos diseñados para la mejora de la interpretabilidad del sistema experto y en las utilidades desarrolladas para la simplificación del mismo.

Los Capítulos 5 y 6 presentan un resumen de las conclusiones y objetivos principales alcanzados en el proyecto y las posibles y más importantes lineas de trabajo futuro a realizar empleando este proyecto como base.

Por último, la sección de Apéndices muestra una ampliación de la información expuesta en el Capítulo 4 referente a la algoritmia desarrollada y el desarrollo de gráficos desplazados aquí para conseguir una mayor resolución de cara al lector.

Capítulo 2

LÓGICA DIFUSA

2.1. ¿Qué es la *Lógica Difusa*?

Para poner en situación al lector resumiremos el concepto de lógica difusa con las siguientes palabras: *la lógica difusa es una lógica multivalente, en oposición a la lógica clásica bivalente, que permite trabajar con conceptos de incertidumbre y vaguedad semántica humanas como mucho, poco o algo.*

Es decir, empleando lógica bivalente un ordenador sólo sería capaz de comprender que, si siendo el subconjunto de personas con estatura mayor o igual a 2 m las que reciben el calificativo *alto*, una persona con 1,8 m de altura reciba el calificativo *bajo*.

Mediante la lógica difusa podremos, **no** aplicar un único valor verdadero a un único subconjunto, sino asignar un determinado valor de pertenencia a cada uno de los subconjuntos definidos. Aplicando esto al ejemplo anterior, podemos decir a modo de explicación que, si siendo los mayores a 2 m altos y los menores a 1,5 m bajos, una persona que mida 1,75 m pertenecerá, aproximadamente (ya que se depende de la definición de las funciones de pertenencia), por igual a ambos subconjuntos *alto* y *bajo*.

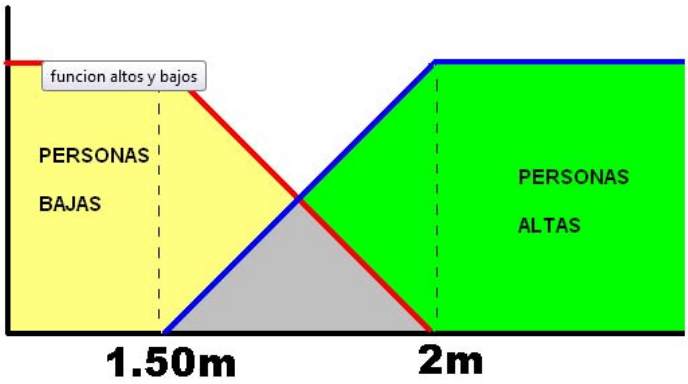

Figura 2.1: Función difusa alto-bajo.

Conviene destacar que lo que consideramos como difuso, impreciso, vago o borroso no es la lógica en sí misma, sino el objeto que estudia, es decir, expresa la falta de definición (nuestra incertidumbre) del concepto a la que se aplica.

Como ya comentaremos después, una de las mayores virtudes del empleo de la unión entre los sistemas expertos y la lógica difusa es la sencillez con la que un usuario no experto en la materia puede emplear el sistema necesitando únicamente emplear criterios comunes de incertidumbre sintáctica, es decir, calificativos como *mucho* o *poco*.

2.2. Antecedentes

El concepto de *lógica difusa* fue concebido a mediados de los años sesenta por **Lofti Zadeh** (figura 2.2), ingeniero eléctrico iraní y profesor de la Universiad de California (Berkeley), quien en 1965 publica el primer artículo de lógica difusa llamado *"Fuzzy Sets"* [Zad65], donde se dan a conocer por primera vez los conceptos de esta técnica.

La lógica difusa nos capacita para emplear un mecanismo de inferencia que permite simular los procedimientos de raciocinio humano en sistemas basados en el conocimiento. La teoría de la lógica difusa enunciada por Zadeh proporciona un marco matemático que permite modelar el grado de incertidumbre asociado inherentemente a los procesos de razonamiento humano de manera que pueda ser interpretado por un ordenador.

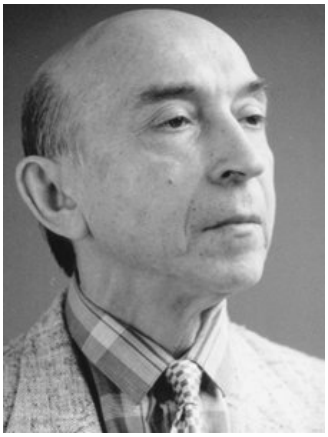

Figura 2.2: Lofti Zadeh.

Esta modelización de la incertidumbre humana proporcionó una nueva herramienta que representaba un salto considerable respecto a la lógica clásica, o bivalente, usada hasta el momento, en la que sólo se puede disponer de dos posibilidades: *verdadero/falso; on/off; 1/0...*

Más tarde, en 1974, **Ebrahim Mamdani** (figura 2.3) aplica los conceptos de lógica difusa en el control de procesos y desarrolla el primer control difuso para la regulación de un motor de vapor [Mam74]. Fue a partir de la obra de Mamdani cuando se empezó a emplear el término lógica difusa como sinónimo de cualquier sistema matemático o computacional que razona con lógica difusa.

Figura 2.3: Ebrahim Mamdani.

En 1985 Takagi y Sugeno aportan a la teoría del control difuso un nuevo método llamado Takagi-Sugeno-Kang (TSK), como alternativa del método Mamdani [TS85]. La principal diferencia entre ambos métodos [HG08] radica en que el modelo de inferencia de Mamdani requiere algún tipo de método para el *desborronamiento* o *defuzzificación*, lo que propicia un alto coste computacional para su resolución. Sin embargo, este coste computacional puede verse reducido empleando una función matemática en el consecuente, tal y como se hace al emplear el modelo de inferencia TSK:

- **Mamdani**: Si x es A y y es B Entonces z es C
- **TSK**: Si x es A y y es B Entonces z es *f(x,y)*

Un **mapa cognitivo** es un grafo dirigido que permite la existencia de retroalimentación entre sus nodos. Cada nodo representa un concepto y la existencia de un arco entre nodos representa una relación causal entre los conceptos representados por dichos nodos. Los nodos toman valores de 0 o 1 y los arcos están definidos por un signo: positivo si se provoca un incremento en el nodo sucesor y negativo si se consigue un decremento. Los mapas cognitivos fueron desarrollados por **Robert Axelrod** [Axe76] para representar sistemas políticos y sociales.

La técnica fue extendida por **Bart Kosko** [Kos86], desarrollando los **mapas cognitivos difusos**, para permitir a los nodos tomar valores en el intervalo continuo [0,1] y sustituir el signo de los arcos por un número en el intervalo [-1,1], que representa la intensidad del la relación entre los nodos conectados.

La unión de los conceptos de lógica difusa desarrollados por Zadeh y el área de control de procesos encuentra numerosas aplicaciones en la industria, medicina, aeronáutica, electrónica, etc.

2.3. Características

El *Principio de Incompatibilidad* [Zad73] dice que la descripción del comportamiento de un sistema complejo no puede realizarse de forma absolutamente precisa. Para solucionar

este problema Zadeh plantea la necesidad de obtener herramientas capaces de manejar de forma rigurosa y fiable información imprecisa, lo cual obliga a desarrollar dos aspectos:

- **Representación de la información imprecisa:** como solución se propone el empleo de la *Teoría de conjuntos difusos*, así como realizar la descripción de la experiencia de los sistemas complejos mediante sus relaciones entrada-salida (proposiciones condicionales del tipo *If-Then*).
- **Inferencia sobre la información imprecisa:** para el proceso de inferencia, es decir, el proceso de cálculo para llegar desde las entradas a las salidas correspondientes, se necesita una forma de combinar la información proporcionada por las entradas y las relaciones condicionales para obtener las salidas. Zadeh estableció la necesidad de un método de inferencia generalizado e introdujo lo que se conoce como *Regla Composicional de Inferencia*.

A partir de este principio, se pueden describir las principales características esenciales de la lógica difusa y los sistemas difusos:

1. El **razonamiento exacto** puede verse como un caso particular del razonamiento aproximado. Cualquier sistema lógico puede ser *fuzzificado* o *emborronado*. Mediante la lógica difusa se puede formular el conocimiento humano de una forma sistemática y puede ser fácilmente incluido en sistemas de ingeniería.
2. El conocimiento se interpreta como una colección de **restricciones difusas** sobre una colección de variables. Los sistemas difusos son especialmente interesantes para la definición de sistemas cuyo modelo exacto es difícil de obtener.
3. La **inferencia** puede verse como un proceso de propagación de estas restricciones difusas.
4. Se utiliza ampliamente en sistemas de **ayuda a la decisión.** La lógica difusa permite obtener decisiones con valores incompletos o información inexacta.

Los sistemas difusos son muy recomendables en aquellos problemas muy complejos donde no existe un modelo matemático simple asociado, así como en procesos que obedecen a un comportamiento **no lineal**. La solución difusa requiere que el conocimiento experto sea expresado lingüísticamente, requisito que es normalmente fácil de satisfacer.

2.4. Sistemas Expertos

En inteligencia artificia (IA), un **sistema experto** (SE) es un sistema computacional que emula la capacidad de tomar decisiones de un humano experto en una temática concreta [Jac86].

Por definirlo de una forma sencilla, podríamos decir que un sistema experto es un *"volcado"* del conocimiento experto de un humano sobre un ordenador. No sería adecuado pensar en un sistema experto como un sustituto para un ser humano, sino como un **apoyo** o un **asistente** para la persona experta en la materia estudiada.

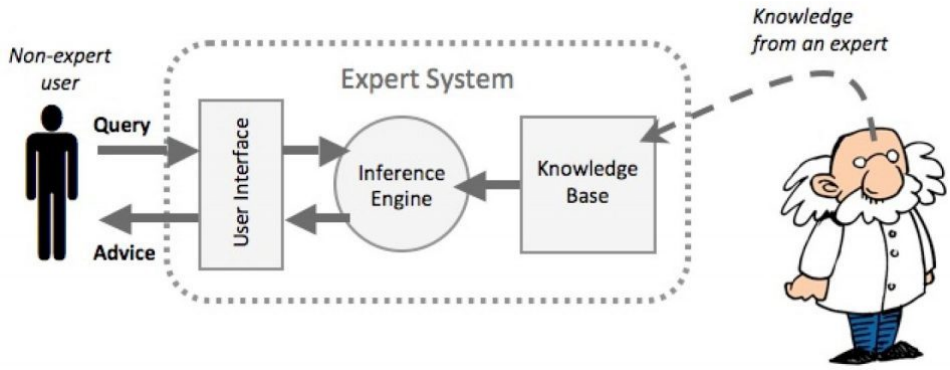

Figura 2.4: Esquema de diseño y empleo de un sistema experto.

El hecho de que sea posible desarrollar un SE para una tarea particular no significa que sea deseable hacerlo. Justificar la creación de un SE puede hacerse de diversas formas, entre las que se encuentran las que se citan a continuación [DGdAM01]:

- En primer lugar, cuando la toma de decisión del experto debe hacerse en entornos peligrosos u hostiles, tales como plantas nucleares, estaciones espaciales, etc.
- También estaría justificado cuando los expertos humanos son incapaces de hacer un trabajo o no pueden usarse por su escasez. A menudo, los expertos humanos escasean y, por consiguiente, su demanda es muy alta y su coste elevado.
- Cuando una experiencia relevante y significativa se está perdiendo en una organización debido a cambios de personal como traslados, jubilaciones, etc., se justifica la

creación de un SE que se pueda convertir en una especie de memoria institucional de la empresa.

- Otra forma de justificarlo sería mediante una alta tasa de recuperación de la inversión.
- Por último, cuando no es posible utilizar otras soluciones alternativas distintas de las que proporciona la inteligencia artificial en general.

Podemos diferenciar entre los tres tipos más comunes de sistemas expertos:

- **Basados en reglas previamente establecidas:** cuya resolución se consigue aplicando reglas heurísticas apoyadas generalmente en lógica difusa. Cabe destacar que las reglas deben cumplir unos criterios de coherencia, de forma que dos reglas distintas no contradigan su posible solución. Este tipo es en el que nos concentramos de aquí en adelante.
    - **Basados en optimización:** como caso particular de los sistemas basados en reglas podemos mencionar los basados en optimización. Estos sistemas se emplean con el objetivo de *calcular* el valor de dichos nodos iniciales (**función objetivo**) para una salida concreta fijada, empleando un conjunto de reglas como **restricciones**. Tanto la función objetivo como las restricciones se emplean para resolver un problema de optimización. Estos sistemas son los que marcan la tendencia actual en el desarrollo de nuevos sistemas expertos.
- **Basados en casos o CBR (Case Based Reasoning):** donde la solución a un problema similar planteado con anterioridad se adapta al nuevo problema [Sch83]. Este tipo de sistemas basa su funcionamiento en experiencias anteriormente vividas, ya sea por el propio sistema o bien por la persona experta, y a partir de este conocimiento de vivencias realizar una asociación con estas experiencias para extraer una solución al nuevo problema. Este tipo de sistemas expertos es el que más se asemeja al proceso de razonamiento humano.
  Como ejemplo de este razonamiento podemos imagina a un niño pequeño que toca una olla caliente y se quema, lo que le proporciona una determinada experiencia. Si ese niño ve una olla caliente otro dia sabrá que no debe tocarla gracias a la experiencia anterior.

- **Basados en redes bayesianas:** con solución basada en la estadística y el teorema de Bayes (expresión 2.1) [VV98].

Sea $A_1, A_2, ..., A_i, ..., A_n$ un conjunto de sucesos mutuamente excluyentes y exhaustivos, y tales que la probabilidad de cada uno de ellos es distinta de cero. Sea B un suceso cualquiera del que se conocen las probabilidades condicionales $P(B|A_i)$. Entonces, la probabilidad $P(A_i|B)$ viene dada por la expresión:

$$P(A_i|B) = \frac{P(B|A_i) \cdot P(A_i)}{P(B)} \tag{2.1}$$

donde:

- $P(A_i)$ es la probabilidad a priori.
- $P(B|A_i)$ es la probabilidad de B en la hipótesis A_i.
- $P(A_i|B)$ es la probabilidad a posteriori.

Las principales ventajas de los sistemas expertos son las siguientes:

- Un sistema experto no sufre **pérdida de facultades** con el paso del tiempo, como sí pasa con un ser humano.
- Al igual que en el caso anterior, un sistema experto no se ve afectado por **condiciones externas** (cansancio, presión...)
- Un sistema experto puede obtener información de una **base de datos** y realizar cálculos numéricos a mayor velocidad que una persona.
- El **coste** de un sistema experto, a la larga, es menor que el coste de emplear a una persona experta durante el mismo tiempo, pese a que la inversión inicial sea mucho mayor [DGdAM01].

Por contra, algunos de los inconvenientes o limitaciones de los sistemas expertos son:

- **Sentido común:** un sistema experto básico no es capaz de discernir si algún dato o resultado no tiene sentido, como sí es capaz de hacer una persona. Por lo

tanto se requeriría un sistema más complejo o el empleo de un sistema **FDIR**, es decir, *Faul detection, isolation, and recovery* (Detección de errores, aislamiento y recuperación).

- Un sistema experto no es capaz de **aprender** por sí mismo, por lo que necesita disponer de un proceso de revisión continua, tal como el *feedback* de un experto para el reajuste de los parámetros del sistema.
- Un sistema experto no es capaz de manejar conocimiento **poco estructurado**.

Algunos de los sistemas expertos más empleados a lo largo de la historia son los siguientes:

2.4.1. Dendral

Dendral, un *portmanteau* de *Dentritic Algorithm*, fue un proyecto pionero en inteligenica artifial en los años sesenta, así como el propio sistema experto producto del proyecto, diseñado por E. Feigenbaum, B. Buchanan, J. Lederberg y C. Djerassi, de la Universidad de Stanford [LBFL80].

Su propósito fundamental era el estudio y la formación de hipótesis en el ámbito de la investigación científica. El objetivo específico del sistema experto era servir de ayuda a los químicos orgánicos para **identificar moléculas orgánicas desconocidas**, a través del análisis de su espectro de masa.

Fue el primer sistema experto en ser utilizado para propósitos reales, al margen de la investigación computacional. Su filosofía se aleja de las estructuras clásicas de los sistemas expertos más típicos, ya que su implementación no separaba de forma explícita el conocimiento del motor de inferencia. Sin embargo, pronto se convirtió en uno de los modelos a seguir por muchos de los programadores de sistemas expertos de la época.

2.4.2. Mycin

Mycin es un sistema experto desarrollado a principios de los años setenta por E. Shortliffe, en la Universidad de Stanford. Estaba inicialmente inspirado en Dendral, pero su principal función consistía en el diagnóstico de **enfermedades infecciosas de la sangre** [SDA+75].

Como novedad, Mycin presentaba la capacidad de *"razonar"* el proceso seguido para llegar a estos diagnósticos y de recetar medicaciones personalizadas a cada paciente: su funcionamiento se basaba principalmente en un sencillo motor de inferencia que manejaba una base de conocimiento de aproximadamente unas 500 reglas.

Tras el proceso de inferencia Mycin mostraba unos resultados de salida consistentes en una serie de posibles enfermedades, ordenadas según su probabilidad asociada, la explicación del por qué de cada uno de estos diagnósticos, y una serie de recomendaciones sobre el tratamento a seguir por el paciente.

Su tasa de aciertos rondaba el 65 %, tasa superior a la de los médicos no especializados, pero inferior al 80 % de acierto de los médicos expertos en la materia. Fue de los primeros sistemas expertos ante los que se mostraron discrepancias éticas y legales entre los usuarios, al volcar la responsabilidad de la salud de una persona a una máquina.

2.4.3. XCon

Originalmente llamado R1, **XCon** (e**X**pert **CON**figurer) era un sistema de producción basado en reglas escrito por J.P. McDermott, de la Universidad Carnegie Mellon, en 1978 para asistir a los pedidos de los sistemas de computadores VAX de DEC, *Digital Equipment Corporation*, seleccionando los componentes del sistema de acuerdo a los requerimientos del cliente [BOBS89].

XCon tenía alrededor de 2.500 reglas justo en su nacimiento. Para 1986, el sistema había procesado 80.000 órdenes y alcanzaba un 97 % de precisión. Las estimaciones in-

ternas de la empresa aproximaban el ahorro generado a 25 millones de dólares, gracias a reducir la necesidad de dar a los clientes componentes gratuitos cuando los técnicos cometían errores, aumentando la velocidad del proceso de ensamblaje e incrementando la satisfacción final del cliente.

2.4.4. Watson

Watson es, actualmente, la joya de la corona del gigante IBM. No es un sistema experto en sí mismo, pero merece la pena ser mencionado junto a los anteriores debido a que representa el futuro de la computación cognitiva [FBCC+10].

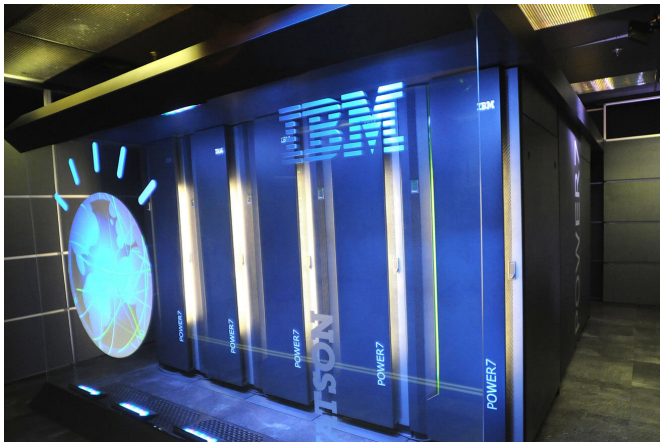

Figura 2.5: Superordenador Watson, IBM.

Watson es una inteligencia artificial capaz de responder a preguntas formuladas en lenguaje natural, que forma parte del proyecto del equipo de investigación *DeepQA*, liderado por D. Ferrucci. La IA se ayuda de una gran variedad de algoritmos y sistemas expertos para la realización de un buen análisis de la pregunta introducida como entrada.

Como claro ejemplo del potencial de esta tecnología se pueden mencionar dos casos:

- En 2011, Watson fue capaz de derrotar, sin acceso a internet, únicamente empleando una base de datos previa como base de conocimiento, a los dos mejores concursantes de la historia del popular concurso estadounidense *Jeopardy!* [FLB+13], basado en un simple juego de pistas y respuestas.

- El equipo del centro oncológico de la Universidad de Carolina del Norte, que emplea el Watson Health [Mal13], la unidad de negocio de IBM que aplica las posibilidades de Watson al campo de la salud, lo hace semanalmente para que el sistema asista al equipo en la evaluación de opciones de tratamiento para pacientes con cáncer que no responden a las terapias estándar. Tras dos semanas de enseñanza en literatura médica y procesamiento de documentos, Watson Health había analizado veinticinco millones de documentos.

Capítulo 3

ESTADO INICIAL

Al inicio del trabajo disponemos de una serie de librerías que permiten desarrollar el cálculo objetivo del sistema experto, sin embargo, como ya se ha comentado anteriormente, estas librerías no permiten una representación gráfica que permita discernir, a golpe de vista, el peso de las relaciones internodales en el proceso de inferencia.

El código empleado como base se basa en programación orientada a objetos. Los objetos desarrollados en la tesis de base son *Relation*, *Nodo* y *FuzzyCognitiveMap*, que definen, respectivamente, la información de los arcos internodales, los nodos del sistema y el modelo completo del sistema. Este último es el objeto de más alto nivel del código, haciendo uso a su vez de los otros dos.

Las capacidades de representación gráfica estaban limitadas al método *gen_graph*, del objeto *FuzzyCognitiveMap* perteneciente a la librería del mismo nombre, se permite obtener un grafo que muestra meramente las relaciones internodales del sistema introducido (figura 3.1).

A continuación en este capítulo, presentaremos el caso de estudio sobre el que trabajaremos y desarrollaremos el proceso de cálculo del sistema de cálculo inicial, así como las modificaciones realizadas para nuestro proyecto, mostrando las utilidades desarrolladas de cara a la simplificación e interepretación del trabajo del usuario en el capítulo siguiente.

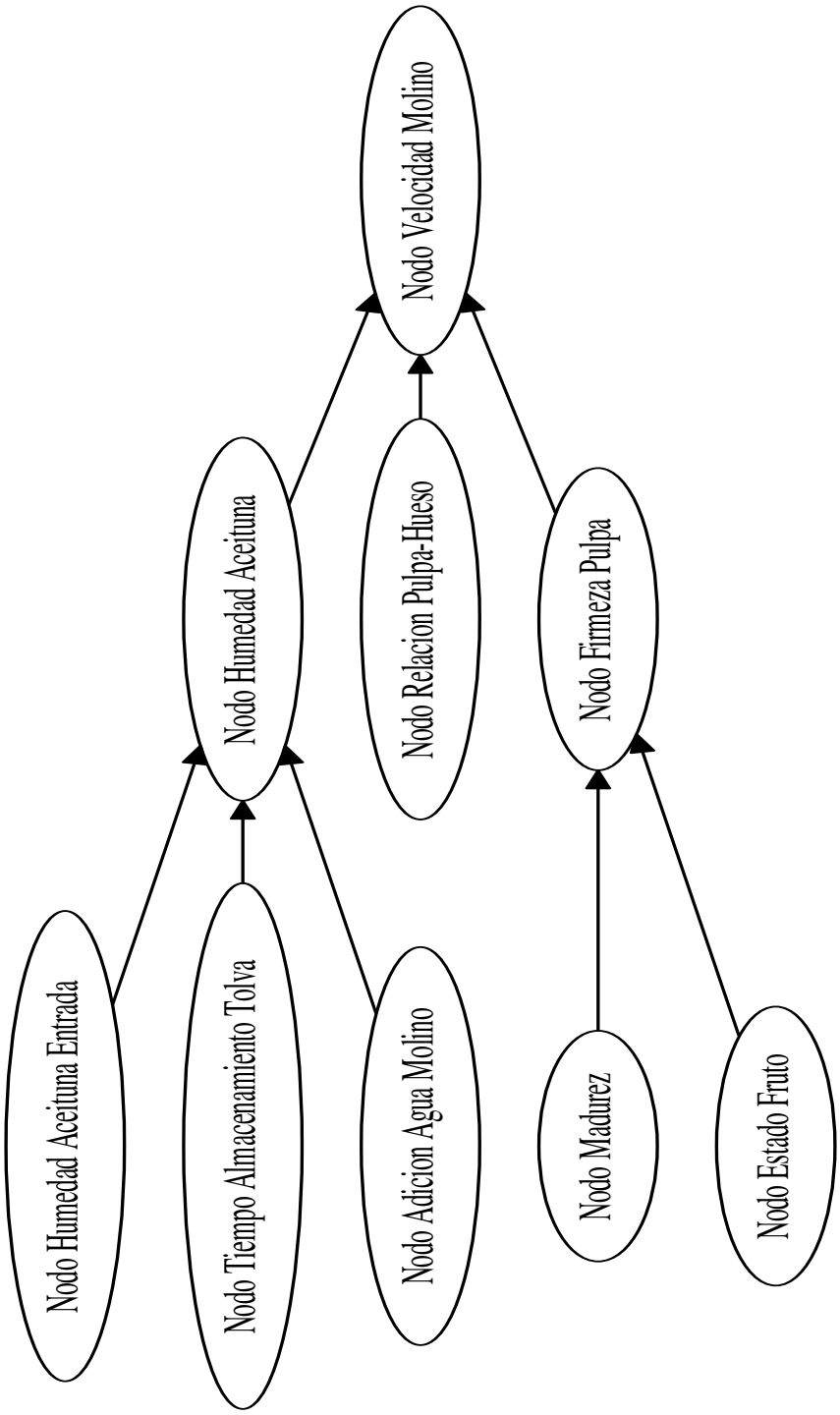

Figura 3.1: Representación gráfica de un sistema generado por el método *gen_graph*.

3.1. Caso de estudio: proceso de elaboración de aceite de oliva virgen (PEAOV)

El caso de estudio sobre el que desarrollaremos la explicación de las utilidades implementadas en el sistema es el del **proceso de elaboración de aceite de oliva virgen (PEAOV)**. El PEAOV es un proceso industrial complejo, con una gran cantidad de variables a tratar y unos objetivos, también tratados a lo largo del presente trabajo como variables de salida, que tienden a presentar valores deseables contrapuestos. Habrá de tenerse en cuenta que en el PEAOV es de vital importancia el correcto balanceamiento entre los conceptos de *cantidad* y *calidad*, aumentando el primero conforme diminuye el segundo y/o avanza la campaña de recolección [Mar15].

A continuación presentaremos dos subsistemas que pueden establecerse dentro de todo el proceso a través de la definición de valores nítidos predefinidos de los nodos y la explicación de las relaciones que pueden establecerse entre ellos.

En primera instancia mostramos el subsistema **Separación de la interfase sólido-líquido** (figura 3.2), consistente en el modelo empleado para definir la interfaz de separación en el decanter de aceite, pudiendo estimar valores tales como la limpieza final o el agotamiento del aceite.

Sin embargo, el subsistema principal que será objeto de nuestro estudio es el de **Preparación de la pasta** (figura 3.3). Mediante el modelo de preparación de la pasta obtendremos las salidas referentes a los valores de *defecto de la aceituna*, *nivel de frutado de la pasta* y *estado de batido de la pasta*. Para todos los casos siempre habrá de tenerse en cuenta la diferencia entre parámetros del proceso y variables de decisión:

- **Parámetros del proceso:** son variables del PEAOV que se consideran fijadas previamente al inicio del cálculo del modelo, es decir, no son controlables en ningún caso. Como ejemplo podemos mencionar la humedad de la aceituna a la entrada de la tolva o la madurez de la misma, parámetros que dependerán del proceso y momento de recogida de la aceituna durante la campaña de recolección.

- **Variables de decisión:** son variables del PEAOV que deben ser especificados al inicio del cálculo del modelo, pudiendo ser controlados por parte del usuario del sistema. En el caso de sistemas expertos basados en optimización estas variables serán la salida objetivo del cálculo, de forma que se establezcan unas variables objetivo óptimas determinadas.

Para mayor simplicidad, desde este punto trabajaremos con unos valores nítidos fijos para los nodos iniciales (tabla 3.1), teniendo en cuenta que el universo del discurso de nuestro sistema $U(v_i) \in [1, 5]$, representando estos valores, como ya apuntamos anteriormente, términos difusos como *muy bajo*, *normal* o *muy bueno*, en función del nodo tratado en cada momento, teniendo en cuenta que emplearemos valores aproximados a los calculados para los nodos intermedios en caso de necesitar emplearlos como iniciales en alguna simplificación.

Preparación de la pasta	
Adición Agua Batidora	2
Adición Agua Molino	1.5
Adición Coadyuvantes	3.5
Desgaste Criba	2.5
Desgaste Martillos	1
Enfermedad Aceituna	3
Estado Fruto Entrada	2.5
Flujo Entrada Molino	5
Humedad Aceituna Entrada	1
Madurez	4
Relación Pulpa-Hueso	1
Suciedad	2
Tamaño Criba	2
Temperatura Batido	5
Tiempo Almacenamiento Tolva	4
Tiempo Batido	3.5
Tipo Criba	4.5
Separación de interfase	
Adición Agua Decanter	2
Contenido Aceite Pasta	2
Estado Batido	$*^1$
Humedad Pasta	$*^2$
Posición Presillas	1
Ritmo Producción	5
Temperatura Batido	$*^3$
Velocidad Diferencial	2
Velocidad Principal	2

Tabla 3.1: Valores nítidos de los nodos iniciales de los subsistemas.

$*^1$: el nodo Estado Batido es un nodo final del subsistema de preparación de la pasta, por lo que empleremos su valor de salida en caso de necesitarlo como variable de decisión para el subsistema de separación de la interfase. Su valor será 4.002.

$*^2$: el nodo Humedad Pasta es un nodo intermedio del subsistema de preparación de la pasta, por lo que emplearemos su valor de salida en caso de necesitarlo como variable de decisión para el subsistema de separación de la interfase. Su valor será 1.962.

$*^3$: el nodo Temperatura de Batido ya viene fijado en el subsistema de preparación de la pasta como un nodo inicial, por lo que emplearemos el mismo valor, es decir, 5.

Las relaciones que pueden establecerse entre los nodos de los subsistemas son las que determinan el contenido de las matrices de relación causal propiedad de los arcos de los grafos. Los diferentes tipos de relación se exponen a continuación:

- **Relaciones bivalentes**: establecen una relación lineal entre el nodo precedente y el nodo sucesor, de forma que si la relación es positiva el valor del nodo sucesor aumentará proporcionalmente al del nodo predecesor y viceversa. Ejemplos de matrices que determinan este tipo de relaciones son las mostradas a continuación:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

- **Relaciones univalentes**: establecen una relación asimétrica, de forma que no todos los valores del nodo predecesor influyen de la misma forma sobre el nodo sucesor. Además, la influencia del nodo predecesor siempre es del mismo signo (positiva o negativa) sobre el nodo sucesor. Ejemplos de matrices que determinan este tipo de relaciones son las mostradas a continuación:

$$\begin{pmatrix} 0 & 0{,}25 & 0{,}5 & 0{,}75 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0{,}25 & 0{,}5 & 0{,}75 & 1 \end{pmatrix}.$$

Estas matrices determinan una influencia positiva sobre el primer valor y sobre el quinto valor del universo del discurso, respectivamente, debido a que sus valores son crecientes hacia la derecha. Las matrices siguientes muestran relaciones univalentes contrarias a las anteriores, es decir, representan una influencia negativa sobre los mismos valores del universo del discurso que las anteriores:

$$\begin{pmatrix} 1 & 0{,}75 & 0{,}5 & 0{,}25 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0{,}75 & 0{,}5 & 0{,}25 & 0 \end{pmatrix}.$$

- **Relaciones de punto óptimo** o ***sweet-spot***: un cierto valor del nodo predecesor origina el máximo o mínimo valor del nodo sucesor. Matrices que ejemplifican relaciones de punto óptimo mínimo y máximo son las siguientes:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

  Mediante este tipo de relaciones pueden expresarse fácilmente efectos de saturación de un nodo predecesor sobre un sucesor.

Ejemplos de estas relaciones aplicadas a un caso práctico son las que ya mostramos en la figura 3.4. La relación que se establece entre los nodos *Humedad Aceituna Entrada* y *Humedad Aceituna* es bivalente positiva, o simplemente bivalente, dado que un alto (bajo) valor del primero propicia un alto (bajo) valor del segundo. Por el contrario, la relación establecida entre los nodos *Adicion Agua Molino* y *Humedad Aceituna* es univalente incremental, de forma que un alto valor de adición de agua al molino tenderá

a incrementar el valor de la humedad en la aceituna. Sin embargo, como se explicará más en detalle en la siguiente sección, un bajo valor de adición de agua al molino no contribuye o lo hace en muy pequeña medida sobre la humedad de la aceituna.

A la vista de este ejemplo, se hace patente que un nodo no puede tener predecesores tales que sus relaciones sean únicamente univalentes, debido a que podrían dejar el nodo sucesor indefinido por tener una incidencia nula sobre el mismo.

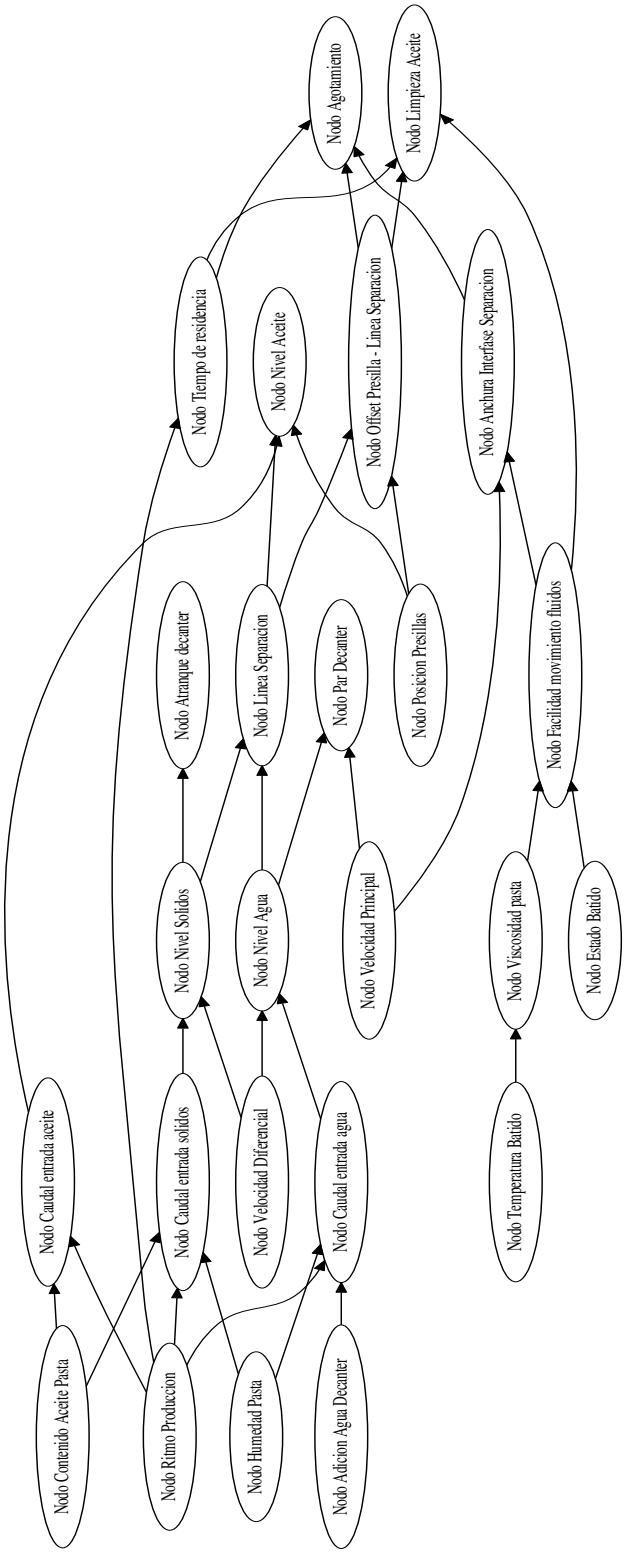

Figura 3.2: Modelo de separación sólido-líquido.

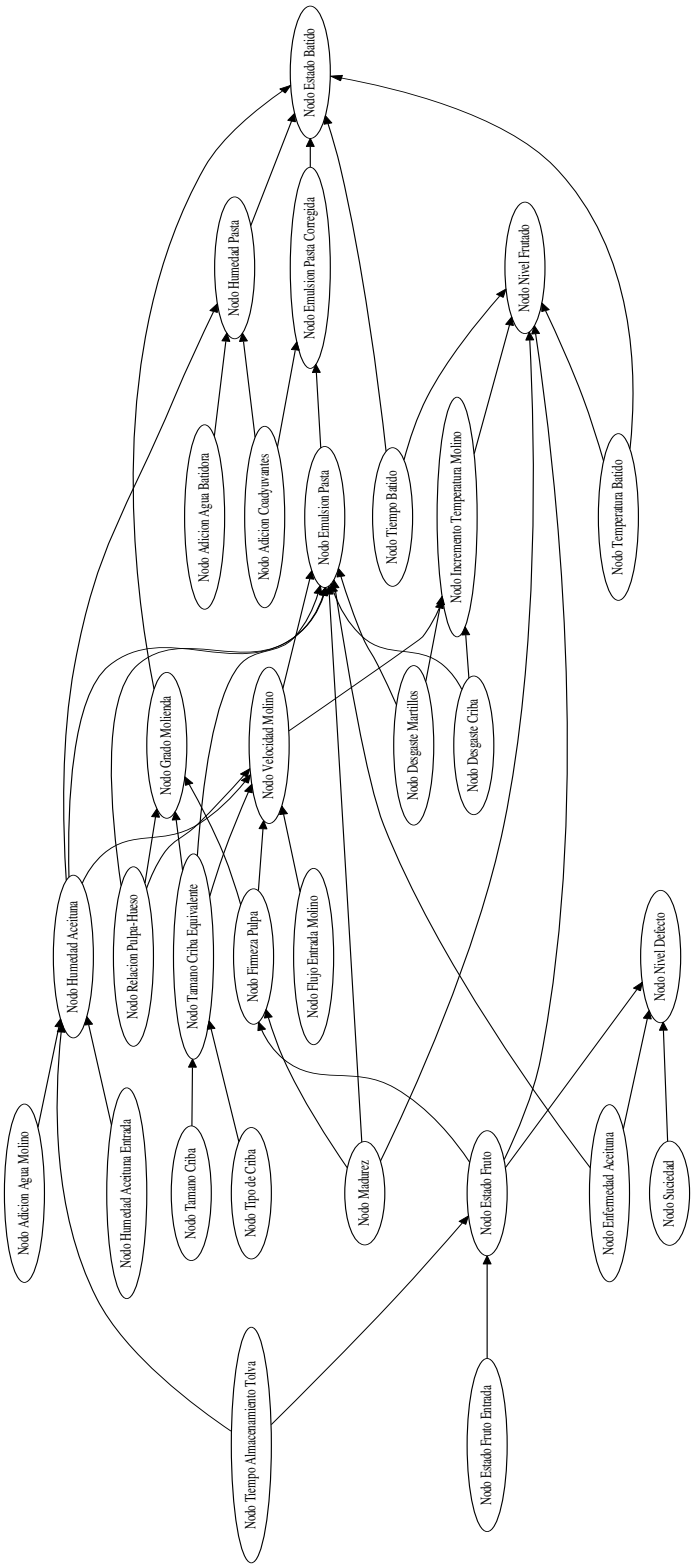

Figura 3.3: Modelo de preparación de la pasta.

3.2. Proceso de cálculo

Los Mapas Borrosos Cognitivos (MBC) representan un enfoque de modelado muy conveniente para sistemas con un alto número de variables. Esta metodología facilita la obtención del conocimeinto de los expertos y a su vez permite utilizar un enfoque sistemático en dicha obtención [MWGO16].

El método de modelado propuesto se basa en Redes Cognitivas Dinámicas Simplificadas (sDCN). Formalmente, el modelo se define como una tupla:

$$\mathbf{M} = \langle \mathbf{V}, \mathbf{A} \rangle, \tag{3.1}$$

donde $\mathbf{V}$ representa el conjunto de nodos - representando variables -, y $\mathbf{A}$ representa el conjunto de arcos - representando las relaciones establecidas entre las variables.

Los nodos de la red pueden actuar como nodos predecesores o como sucesores, en función del rol que cumplan en el proceso de computación. Los nodos predecesores son parecidos a las variables incluidas en la parte antecedente de una regla difusa, mientras que los nodos sucesores se corresponderían con las variables en la parte subsecuente a dicha regla. Para modelos multicapa, un nodo puede actuar siguiendo ambos roles simultáneamente, siempre y cuando haya arcos tanto de entrada al nodo como de salida del mismo.

Para cada nodo $v_i \in \mathbf{V}$ de la red se definen las siguientes propiedades:

- U_{v_i}: el universo del discurso del nodo. Representa el conjunto que contiene todos los posibles valores nítidos (*crisp values*) de la variable representada por v_i.
- H_{v_i}: representa la colección de términos (conjuntos difusos) $L_{v_i}^k$ definidos en U_{v_i}:

$$L_{v_i}^k = \{\langle x, \mu_{L_{v_i}^k}(x)\rangle : x \in U_{v_i}\}, \tag{3.2}$$

$$H_{v_i} = \{L_{v_i}^k, k = 1, 2, ..., K_i\}. \tag{3.3}$$

Aquí, K_i representa el número total de conjuntos definidos para el nodo.

- $S_f(v_i)$: vector que contiene el grado de pertenencia a cada conjunto difuso L_{v_i} de un v_i dado para una entrada nítida.

$$S_f(v_i) = [\mu_{L^1_{v_i}}, ..., \mu_{L^k_{v_i}}]^T. \tag{3.4}$$

- $S_c(v_i)$: el valor nítido asignado a cada nodo. Si v_i actúa como nodo sucesor, este valor se calcula usando el método explicado debajo en las ecuaciones (3.5) - (3.7). Si v_i actúa como nodo predecesor, este valor se considera ya disponible, ya sea obtenido como una entrada externa del nodo - si el nodo es un nodo inicial - o habiendo sido previamente calculado.

Por otro lado, para cada arco a_{ij} se definen las siguientes propiedades:

- ω_{ij}: intensidad de la relación entre los nodos v_i y v_j.
- R_{ij}: matriz de relación causal. Los valores de la matriz deben ser mayores o iguales a 0 y definen la relación entre los conjuntos difusos de los nodos predecesor y sucesor conectados por el arco. El tamaño de la matriz es $K_i \times K_j$, siendo K_i y K_j el número de términos en H_{v_i} y H_{v_j} respectivamente.

Estas propiedades de los arcos ω_{ij} y R_{ij} juegan un papel similar a los de las reglas en los sistemas de lógica difusa basados en reglas, ya que estos elementos engloban la relación entre las diferentes variables del sistema. En la matriz R_{ij} cada fila se asocia con un conjunto difuso definido en U_{v_i}, y cada columna se asocia con un conjunto difuso definido en U_{v_j}. Cada dato de R_{ij} puede ser visto como una regla que relaciona el valor del predecesor (nodo v_j) con el sucesor (nodo v_i), donde el peso de la regla viene dado por el producto vectorial de cada valor de la matriz por el peso asociado al arco. Por ejemplo, siendo $L^k_{v_i}$ un conjunto difuso triangular y denotando $q_i = [q_i^1\ q_i^2\ ...\ q_i^{K_i}]^T$ los picos de estos conjuntos, cada entrada de la matriz se asocia a una regla del tipo:

$$Si\ v_j\ es\ L^k_{v_j}\ Entonces\ v_j\ es\ q_i^a,\ con\ peso\ \omega_{ij}R^{ab}_{ij}.$$

Como ejemplo, consideramos la siguiente matriz:

$$R_{ij} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Las dimensiones de la matriz nos indica que en total han sido definidos tres conjuntos difusos tanto para U_{v_i} como para U_{v_j}, y las reglas asociadas serán:

$$\begin{array}{l} Si\ v_j\ es\ L^1_{v_j}\ Entonces\ v_i\ es\ q^3_i,\ con\ peso\ \omega_{ij}. \\ Si\ v_j\ es\ L^2_{v_j}\ Entonces\ v_i\ es\ q^2_i,\ con\ peso\ \omega_{ij}. \\ Si\ v_j\ es\ L^3_{v_j}\ Entonces\ v_i\ es\ q^1_i,\ con\ peso\ \omega_{ij}. \end{array}$$

El cálculo del valor de un nodo sucesor, dados los valores de sus predecesores, se calcula de la siguiente forma:

1) El impacto recibido por el nodo i se define como:

$$w_i = \sum_{j=1}^{n_i} \omega_{ij} R_{ij} S_f(v_j) = [w_i^1\ w_i^2\ ...\ w_i^{K_i}]^T \tag{3.5}$$

2) El cálculo del valor nítido $S_c(v_i)$ del nodo se calcula como una media ponderada del peso que tiene cada uno de los picos del conjunto sobre el nodo:

$$S_c(v_i) = \frac{\sum_{k=1}^{K_i} w_i^k\ q_i^k}{\sum_{k=1}^{K_i} w_i^k} \tag{3.6}$$

3) Finalmente, podemos obtener el vector de pertenencia de la siguiente forma:

$$S_f(v_i) = [\mu_{L^1_{v_i}}(S_c(v_i))\ \mu_{L^2_{v_i}}(S_c(v_i)), ..., \mu_{L^{K_i}_{v_i}}]^T. \tag{3.7}$$

La ecuaión (3.6) realiza simultáneamente los pasos de inferencia y *desborronamiento*, y es muy similar al cálculo del valor nítido, mediante el modelo de orden cero de un TSK, a la salida de un conjunto de reglas.

3.2.1. Ejemplo práctico de cálculo

A continuación mostraremos un ejemplo práctico de las ecuaciones anteriores aplicado a un pequeño subgrafo del modelo completo (figura 3.4):

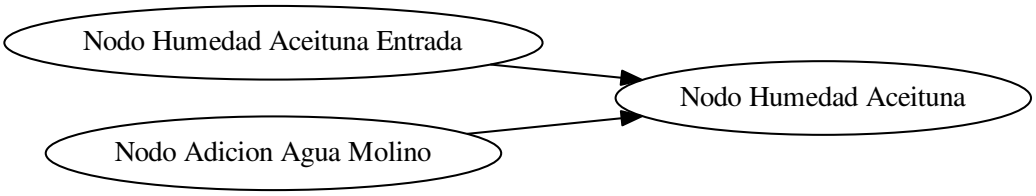

Figura 3.4: Subgrafo para ejemplo de cálculo de valor nítido.

Llamaremos a los nodos *Humedad Aceituna Entrada* y *Adicion Agua Molino* como v_1 y v_2, respectivamente, siendo el nodo objetivo del cálculo, *Humedad Aceituna* , nombrado como v_3.

Por lo tanto, partimos de los siguientes datos iniciales:

- Intensidad de las relaciones:
  - $\omega_{13} = 0.75$
  - $\omega_{23} = 0.5$
- Matrices de relación causal:
  - $$R_{13} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- $$R_{23} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0{,}25 & 0{,}5 & 0{,}75 & 1 \end{pmatrix}$$

  A la vista de esta matrices podemos apuntar algo: la segunda matriz tiene una columna completa de ceros. Si el nodo precedente de dicha matriz tiene un valor no negativo en la misma posición que dicha columna de ceros, al menos ese valor no contribuirá al valor nítido final del nodo sucesor. Esto lo veremos ejemplificado a continuación.

- Valores nítidos de los nodos iniciales:

  - $S_c(1) = 1$
  - $S_c(2) = 1.5$

Una vez establecidos los datos de partida que el algoritmo necesita para el cálculo de cada nodo sucesor vamos a comenzar con el proceso de cálculo. Según (3.5):

$$w_3 = \omega_{13} R_{13} S_f(v_1) + \omega_{23} R_{23} S_f(v_2) = [w_3^1\ w_3^2\ w_3^3\ w_3^4\ w_3^5]^T \tag{3.8}$$

Por lo que sustituyendo valores quedará:

$$w_3 = 0{,}75 \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot [1\ 0\ 0\ 0\ 0]^T +$$

$$+ 0{,}5 \cdot \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0{,}25 & 0{,}5 & 0{,}75 & 1 \end{pmatrix} \cdot [0{,}5\ 0{,}5\ 0\ 0\ 0]^T = \tag{3.9}$$

$$= \begin{pmatrix} 0{,}75 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0{,}0625 \end{pmatrix} = \begin{pmatrix} 0{,}75 \\ 0 \\ 0 \\ 0 \\ 0{,}0625 \end{pmatrix}$$

Este es el vector de impacto del nodo *Humedad Aceituna*, por lo que a continuación aplicaremos la expresión (3.6) para realizar la inferencia y la *defuzzyficación* del mismo para acabar obteniendo el valor nítido del nodo.

Como ya se vio anteriormente:

$$S_c(v_3) = \frac{w_3^1\ q_3^1 + w_3^5\ q_3^5}{w_3^1 + w_3^5} \tag{3.10}$$

Y si sustituimos al igual que antes:

$$S_c(v_3) = \frac{0{,}75 \cdot 1 + 0{,}0625 \cdot 5}{0{,}75 + 0{,}0625} = 1{,}308 \tag{3.11}$$

Con lo que obtenemos el valor final nítido del nodo sucesor en cuestión. A la hora de la implementación en software del algoritmo, éste se desarrolla de forma ligeramente distinta para que su comprensión sea más sencilla, concatenando matrices y empleando un factor de normalización para una disminución de operaciones totales cuando haya

mayor cantidad de nodos precedentes. Sin embargo, más adelante comentaremos las variaciones desarrolladas en esta fase del algoritmo para la consecución de los objetivos de este proyecto.

Tal y como apuntamos durante la ejemplificación de las matrices de relación causal, podemos comprobar fácilmente cómo el primer término de $S_f(v_2)$ no contribuye al resultado final, ya que siempre multiplica por una columna completa de ceros en la matriz de relación causal. Esto puede provocar que un nodo predecesor no aporte ningún tipo de contribución a un nodo sucesor, como podría haber pasado aquí de haber tenido $S_c(v_2) = 1$.

3.2.2. Modificación del código de cálculo

Si bien el cálculo teórico del valor nítido de los nodos sucesores se realiza tal y como se ha explicado y ejemplificado a lo largo de este capítulo, se emplean pequeñas modificaciones a la hora de implementarse en el código de programación, de forma que pueda aprovecharse el potencial del lenguaje empleando cálculo vectorial y matricial. Esto se hace patente al emplear como valores de entrada una concatenación tanto de los vectores de pertenencia difusa (3.12) como de las matrices resultantes de multiplicar las matrices de relación causal y las intensidades de las relaciones (3.13) de los nodos precedentes.

El cálculo desarrollado, aplicado al ejemplo anterior, es el siguiente:

$$S_f(v_1|v_2) = [1\ 0\ 0\ 0\ 0\ 0{,}5\ 0{,}5\ 0\ 0\ 0]^T \tag{3.12}$$

$$R_t = [\omega_1 R_1 | \omega_2 R_2] = \begin{pmatrix} 0{,}75 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0{,}75 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0{,}75 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0{,}75 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0{,}75 & 0 & 0{,}125 & 0{,}25 & 0{,}375 & 0{,}5 \end{pmatrix} \tag{3.13}$$

1. En primer lugar calculamos el vector de impacto:

$$w_3 = R_t \cdot S_f(v_1|v_2) = \begin{pmatrix} 0{,}75 \\ 0 \\ 0 \\ 0 \\ 0{,}0625 \end{pmatrix} \tag{3.14}$$

2. A continuación obtenemos el vector de impacto normalizado, dividiendo el vector de impacto entre un factor de normalización que no es más que el sumatorio de los valores obtenidos en (3.14):

$$\bar{w_3} = \frac{w_3}{\sum_k w_3^k = 0{,}8125} = \begin{pmatrix} 0{,}9231 \\ 0 \\ 0 \\ 0 \\ 0{,}0769 \end{pmatrix} \tag{3.15}$$

3. Por último calculamos el valor nítido mediante el producto de (3.15) y el universo del discurso:

$$S_c(v_3) = [1\ 2\ 3\ 4\ 5] \cdot \begin{pmatrix} 0{,}9231 \\ 0 \\ 0 \\ 0 \\ 0{,}0769 \end{pmatrix} = 1{,}308 \tag{3.16}$$

Ahora bien, si bien este método de cálculo es más sencillo y potente que el explicado teóricamente, presenta el inconveniente de que no se puede discernir el peso de los nodos

predecesores individualmente sobre el sucesor y la modificación de código que permite resolver dicho inconveniente es el primer paso a dar para el desarrollo de nuestras utilidades.

Para poder discernir adecuadamente el peso de los nodos predecesores individualmente basta con repetir el cálculo anterior **sin concatenación inicial de matrices.** Mostraremos como ejemplo el cálculo del nodo *Humedad Aceituna Entrada* anterior, sin embargo, para una mejor ejemplificación le asignaremos un valor nítido de 2.5.

$$S_f(v_1) = [0\ 0{,}5\ 0{,}5\ 0\ 0]^T \tag{3.17}$$

$$\bar{R_1} = [\omega_1 R_1] = \begin{pmatrix} 0{,}75 & 0 & 0 & 0 & 0 \\ 0 & 0{,}75 & 0 & 0 & 0 \\ 0 & 0 & 0{,}75 & 0 & 0 \\ 0 & 0 & 0 & 0{,}75 & 0 \\ 0 & 0 & 0 & 0 & 0{,}75 \end{pmatrix} \tag{3.18}$$

1. Al igual que antes calculamos el vector de impacto, pero esta vez será parcial para el nodo en cuestión:

$$w_{3|1} = \bar{R_1} \cdot S_f(v_1) = \begin{pmatrix} 0 \\ 0{,}375 \\ 0{,}375 \\ 0 \\ 0 \end{pmatrix} \tag{3.19}$$

2. A continuación obtenemos el vector de impacto normalizado parcial, que se obtiene de la misma forma que anteriormente: a partir de la suma de los valores obtenidos en 3.19 para cada nodo y sus respectivos vectores de impacto parciales. El factor de

normalizado seguirá siendo 0.8125 al no haber variado las matrices de relación causal:

$$\bar{w}_{3|1} = \frac{w_{3|1}}{0{,}8125} = \begin{pmatrix} 0 \\ 0{,}4615 \\ 0{,}4615 \\ 0 \\ 0 \end{pmatrix} \tag{3.20}$$

3. Por último calculamos el sumatorio del vector obtenido en 3.20, resultado que muestra el peso relativo del nodo calculado sobre el sucesor en cuestión:

$$\hat{W}_{3|1} = \sum_k \bar{w}^k_{3|1} = 0{,}9231 \tag{3.21}$$

Pese a que pueda parecer que el peso relativo es independiente del valor nítido, dado que el resultado en la figura 3.21 es idéntico al que podemos comprobar en la figura 3.16 para el mismo nodo, esto no es cierto. En este caso se ha obtenido el mimo valor debido a la relación bivalente (ecuaciones 3.22 y 3.23) existente entre los dos nodos analizados, pero sí que se apreciaría una consideable diferencia en el caso de extablecerse una relación de tipo distinto (ecuaciones 3.24 y 3.25), como puede verse a continuación para unas matrices simplificadas. Mostramos distintas matrices de relación causal multiplicadas por vectores de pertenencia:

$$\sum \left[\begin{pmatrix} 0{,}75 & 0 & 0 \\ 0 & 0{,}75 & 0 \\ 0 & 0 & 0{,}75 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right] = \sum \begin{bmatrix} 0{,}75 \\ 0 \\ 0 \end{bmatrix} = 0{,}75 \tag{3.22}$$

$$\sum \left[\begin{pmatrix} 0{,}75 & 0 & 0 \\ 0 & 0{,}75 & 0 \\ 0 & 0 & 0{,}75 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0{,}5 \\ 0{,}5 \end{pmatrix} \right] = \sum \begin{bmatrix} 0 \\ 0{,}375 \\ 0{,}375 \end{bmatrix} = 0{,}75 \tag{3.23}$$

$$\sum \left[\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0{,}5 & 0 \\ 0 & 0{,}5 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right] = \sum \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = 0 \tag{3.24}$$

$$\sum \left[\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0{,}5 & 0 \\ 0 & 0{,}5 & 2 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0{,}5 \\ 0{,}5 \end{pmatrix} \right] = \sum \begin{bmatrix} 0 \\ 0{,}25 \\ 1{,}25 \end{bmatrix} = 1{,}5 \tag{3.25}$$

Capítulo 4

ALGORITMIA DESARROLLADA

El propósito inicial del desarrollo de las librerías del sistema experto producto de la tesis que empleamos como base era el de diseñar un sistema de cálculo y soporte, es decir, la prioridad era el cálculo eficiente y correcto de los valores de salida a partir de los valores establecidos como entrada de forma que sirviesen como ayuda al usuario experto. Debido a ello, no se establecieron mecanismos de visualización del motor de inferencia o utilidades secundarias.

En este capítulo vamos a desarrollar la algoritmia implementada en el sistema, que puede ser dividida en dos bloques: un primer **bloque de representación**, que se basará en la correcta diferenciación y representación de cada uno de los nodos del sistema al representarlos gráficamente; y un segundo **bloque de simplificación**, que estará compuesto por una serie de utilidades destinadas a la reducción del sistema estudiado a un nuevo sistema más simple que cumpla unas determinadas características. Todos los algoritmos desarrollados son perfectamente válidos para cualquier sistema diseñado de la misma forma que el estudiado. La documentación referente a los códigos mencionados a lo largo de este capítulo puede encontrarse en el apéndice I.

Estas utilidades se basan en la reducción de nodos y relaciones del sistema, bien a los más importantes o bien a un rango determinado de importancia dentro del total del sistema, apoyándonos en todo momento en el sistema de cálculo modificado en el capítulo anterior y en la mejora visual del sistema de inferencia.

Todos los algoritmos descritos a lo largo de este capítulo, a excepción de la función *gen_bars*, se han programada como métodos del objeto *FuzzyCognitiveMaps*, de forma que el código quede más centralizado en el mismo archivo.

Debemos hacer mención especial al método *visualize_results_graph*, perteneciente al objeto *FuzzyCognitiveMaps*, que es la base de la representación gráfica del sistema, empleando como librería para la creación del modelo NetworkX y Graphviz como motor gráfico mediante un *binding* del mismo, *PyGraphviz*.

La sintaxis de invocación de este método es la siguiente:

```
visualize_results_graph(self,
                        dic_values,
                        nombre_graf,
                        list_nodos_fijos = [],
                        path = None,
                        show_relation_matrices = False,
                        show_node_values = True,
                        show_contribution_values = False,
                        show_bars = False,
                        palette = 'rand',
                        orden_grafo = 'LR',
                        fontsize = 9):
```

donde:

- self: engloba todo lo referente a la clase que estemos empleando, *FuzzyCognitiveMap* en este caso, pudiendo llamar a cualquier atributo o método de la misma.
- dic_values: diccionario que alberga los valores nítidos precalculados de todos los nodos del sistema.
- nombre_graf: nombre que daremos al archivo pdf con el gráfico final.
- list_nodos_fijos: lista de nodos que son considerados parámetros del proceso. Es opcional, y en caso de incluirla su única función es la de representar dichos nodos con forma hexagonal en lugar de elíptica.
- path: ruta del directorio donde guardamos el archivo pdf. Predeterminadamente,

si no se elige ninguna, el propio método asigna como directorio el mismo elegido en la creación de la clase.

- show_relation_matrices: variable que permite determinar si queremos mostrar información en las etiquetas de los arcos internodales. Tenemos tres opciones posibles:

  1. False: valor por defecto. No se muestra ninguna información.
  2. True: muestra las matrices de relación causal multiplicadas por las intensidades de las relaciones.
  3. 'Impact': muestra el porcentaje de impacto normalizado que está teniendo cada nodo predecesor sobre el sucesor en cuestión.

- show_node_values: variable binaria que permite mostrar o no los valores nítidos de los nodos.

- show_contribution_values: variable binaria que permite mostrar o no el vector de impactos $\bar{w}_i$ de cada nodo.

- show_bars: variable binaria que permite controlar la generación o no de gráficos de barras que denotan el impacto de cada nod predecesor del actual. Para más información consultar el epígrafe referente a la función *gen_bars*.

- palette: nombre de la paleta de colores que vamos a emplear en la representación. El valor por defecto es "rand", que realiza una asignación aleatoria de los colores. Para más información consultar el epígrafe referente al método *asign_colors*.

- orden_grafo: sentido de representación de los nodos. Por defecto se representan de izquierda a derecha.

- fontsize: tamaño de fuente predeterminada.

4.1. Bloque de representación

4.1.1. Asignación de colores a nodos

La asignación adecuada de colores a los diferentes nodos del sistema es la piedra angular de todas las utilidades desarrolladas, así como la base de la correcta diferenciación entre los diferentes nodos. La premisa fundamental a seguir en la asignación será que **dos nodos predecesores de un mismo nodo no pueden tener el mismo color**, dado que de esta forma habría confusiones en la representación al emplear las utilidades.

En primera instancia se debe destacar que se han establecido cuatro paletas de colores diferentes que permiten modificar la visualización de los nodos (tablas 4.1, 4.2, 4.3 y 4.4). Además, existe la posibilidad de realizar una asignación aleatoria de colores, con el inconveniente de que no puede asegurarse el cumplimiento de la premisa anteriormente destacada, si bien es cierto que la probabilidad de que dos nodos predecesores compartan el color exacto es de 1 entre $2{,}815 \cdot 10^{14}$, aunque a ojos del usuario colores muy similares puedan ser percibidos como iguales.

Cada una de las paletas cuenta con un color extra al final de la misma, empleado en utilidades explicadas con posterioridad, con el objetivo de servir de indicador del valor nítido de cada nodo. Estas paletas son las mostradas a continuación, donde podemos observar tanto el color como su nombre y su codificación hexadecimal.

Amarillo brillante	FFFF00
Naranja claro brillante	FFA420
Rojo claro anaranjado	F75E25
Amarillo azafrán	F5D033
Naranja salmón	E55137
Rosa claro	EA899A
Naranja brillante	FF2301
Violeta erica	DE4C8A
Rojo violeta	922B3E
Pardo corzo	59351F
Negro	000000

Tabla 4.1: Paleta de colores: colores cálidos.

Azul zafiro	1D1E33
Beige verdoso	BEBD7F
Gris musgo perlado	898176
Azul violeta	354D73
Verde patina	316650
Gris piedra	8B8C7A
Azul brillante	3E5F8A
Verde hierba	35682D
Rojo negruzco	412227
Azul pastel	5D9B9B
Rojo claro brillante	FE0000

Tabla 4.2: Paleta de colores: colores frios.

Naranja brillante	FF2301
Azul brillante	3E5F8A
Amarillo brillante	FFFF00
Verde brillante	00F700
Rojo claro brillante	FE0000
Naranja claro brillante	FFA420
Marrón señales	6C3B2A
Rosa	E63244
Negro	000000

Tabla 4.3: Paleta de colores: colores brillantes.

Pardo verdoso	826C34
Pardo ocre	955F20
Marrón señales	6C3B2A
Pardo arcilla	734222
Pardo cobre	8E402A
Pardo corzo	59351F
Pardo oliva	6F4F28
Pardo nuez	5B3A29
Pardo rojo	592321
Sepia	382C1E
Rojo claro brillante	FE0000

Tabla 4.4: Paleta de colores: colores otoñales.

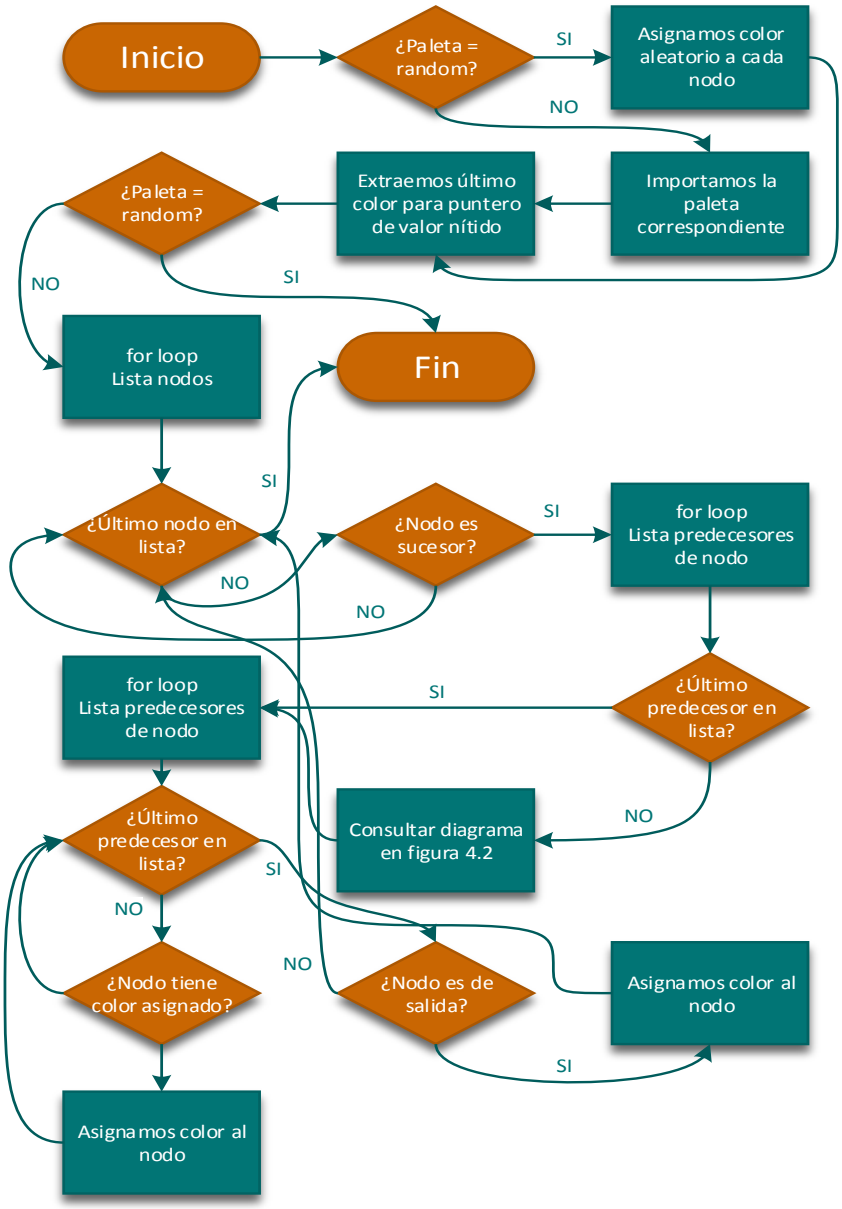

Figura 4.1: Diagrama de flujo del algoritmo de asignación de colores: bloque principal.

La asignación se realiza mediante el método *_asign_colors*, cuyos diagramas de flujo podemos observar en las figuras 4.1 y 4.2. El primero de ellos representa el bloque principal del algoritmo, mientras que el segundo indica los pasos desarrollados cada vez que se estudia el predecesor del nodo estudiado actualmente.

El orden de asignación de los colores se realiza de delante hacia atrás, es decir, no asignamos color al nodo actual en cada iteración del bucle general (a menos que sea un nodo final), sino que asignamos colores a los nodos **precedentes** de dicho nodo. Esta forma de proceder puede causar conflictos que, pese a haber sido solucionados, vamos a comentar a continuación.

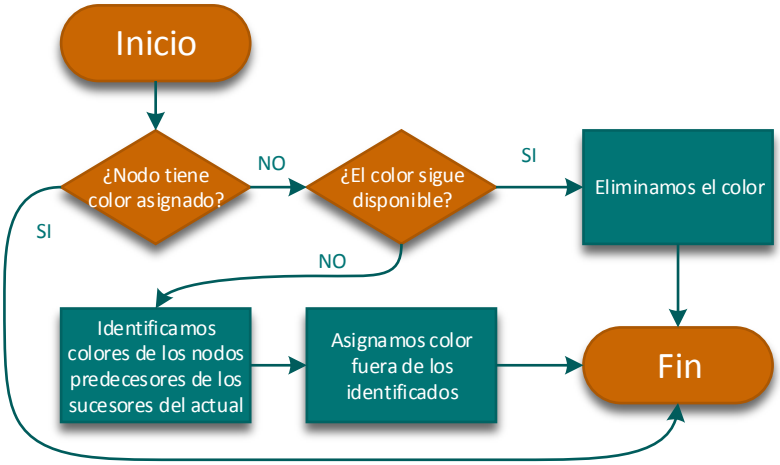

Figura 4.2: Diagrama de flujo del algoritmo de asignación de colores: bloque secudario.

El posible conflicto consiste en la repetición de colores en nodos precedentes de un sucesor común, conflicto resoluble con el paso *Identificamos colores de los nodos predecesores de los sucesores del actual* del la figura 4.2.

El algoritmo siempre intenta asignar el color por orden de lista en la paleta empleada, sin embargo, pueden ocurrir casos como los de la figura 4.3:

1. En primer lugar el algoritmo recorre los *Nodos predecesor 1, 2 y 3*, pero como no son nodos sucesores (nodos intermedios o finales) no se realiza ninguna acción sobre ellos.

2. Una vez el algoritmo alcanza el *Nodo sucesor 1* asigna colores a sus predecesores según el orden de la paleta. Asignará naranja brillante al *Nodo predecesor 1* y azul brillante al *Nodo predecesor 2*.

3. A continuación trabajamos con el *Nodo sucesor 2*, que detecta que el *Nodo predecesor 1* ya tiene color asignado, por lo que elimina el naranja brillante de las posibilidades de asignación. Justo después asignará azul brillante al *Nodo predecesor 3* dado que el naranja brillante no está disponible.

4. Si esta parte del algoritmo no estuviese implementada, una vez se alcanzase *Nodo sucesor 3* detectaría que sus dos nodos precedentes tienen el mismo color y no actuaría en consecuencia. Sin embargo, la acción correctora que realiza el algoritmo consiste en seleccionar el segundo nodo detectado con el mismo color (*Nodo predecesor 3*), y detectar *colores de los nodos predecesores de los sucesores del actual*, es decir:

   - **Nodos sucesores:** *Nodo sucesor 2* y *Nodo sucesor 3*.
   - **Nodos predecesores de los sucesores:** *Nodo predecesor 1*, *Nodo predecesor 2* y *Nodo predecesor 3*.
   - **Colores de los nodos predecesores de los sucesores:** naranja brillante y azul brillante.

5. Por último, se asigna al segundo nodo detectado el primer color disponible fuera de los eliminados en el paso anterior (amarillo brillante) (figura 4.4).

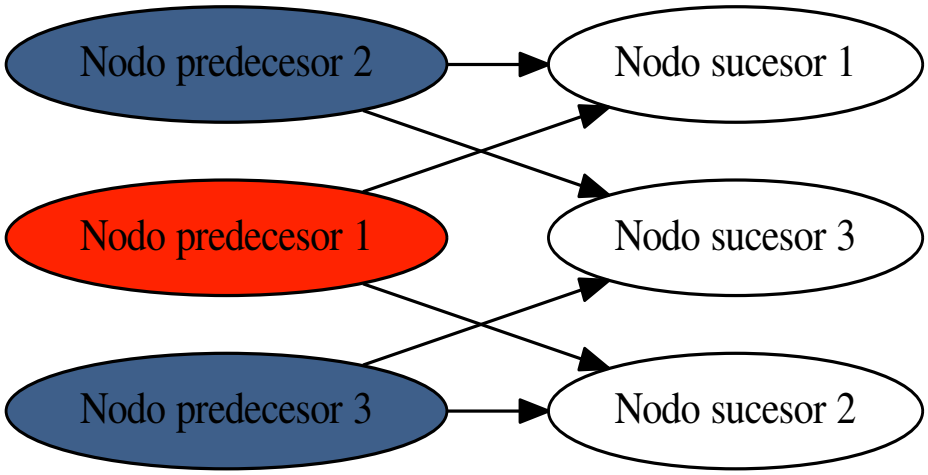

Figura 4.3: Ejemplo de asignación incorrecta de los colores.

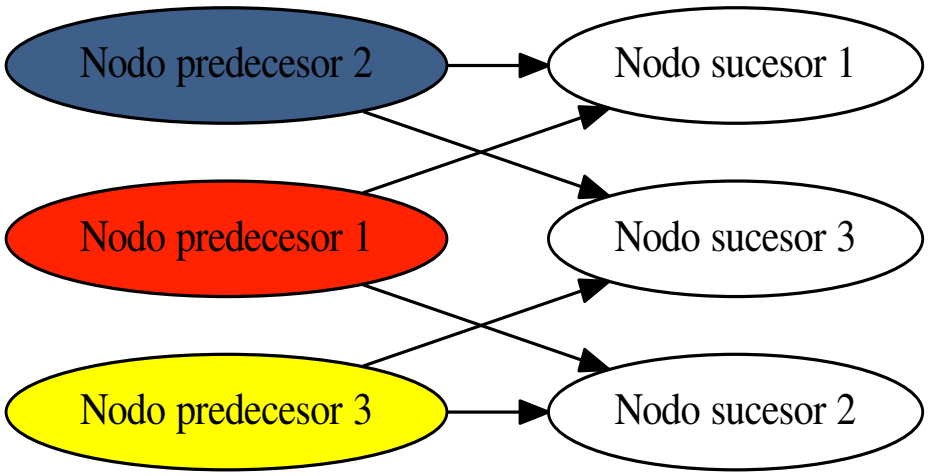

Figura 4.4: Ejemplo de asignación correcta de los colores.

La sintaxis de esta función es la mostrada a continuación:

```
    _asign_colors(self,
                  palette):
```

donde:

- self: engloba todo lo referente a la clase que estemos empleando, *FuzzyCognitiveMap* en este caso, pudiendo llamar a cualquier atributo o método de la misma.
- palette: nombre de la paleta de colores que vamos a emplear en la representación. El valor por defecto es "rand", que realiza una asignación aleatoria de los colores.

4.1.2. Desarrollo de gráficos de barras en nodos sucesores

El empleo de gráficos de barras se fundamenta en la necesidad de la correcta representación del peso relativo de cada nodo precedente sobre sus sucesores. La idea es principalmente que se pueda identificar cuales son los nodos precedentes con más peso rápidamente.

Como primer ejemplo de los gráficos de barras que obtenemos podemos mostrar el siguiente (figura 4.5), correspondiente al nodo *Grado Molienda*. Como se puede comprobar, este gráfico ha sido obtenido empleando la paleta de colores brillantes (tabla 4.3).

Una serie de características deben ser tenidas en cuenta a la vista del resultado obtenido:

- No se inserta título en los gráficos, dado que se insertará posteriormente mediante los atributos de la representación gráfica del propio nodo.
- La escala del gráfico, es decir,el intervalo de valores del eje vertical, depende del valor máximo que alcanzan las barras que representan el peso de los nodos predecesores.
- La leyenda del eje horizontal se refiere a los valores difusos *Very Small*, *Small*, *Medium*, *High* y *Very High*.

- No se incluye leyenda de colores ya que no es necesaria gracias a la fácil diferenciación de colores de los nodos, como veremos dentro de poco.

El diagrama de flujo del algoritmo desarrollado es el mostrado en la figura 4.6:

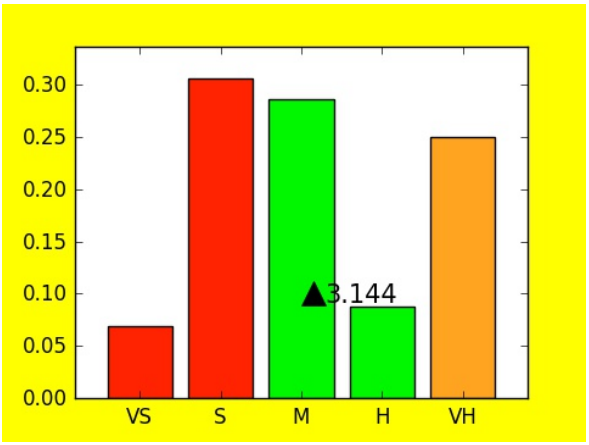

Figura 4.5: Gráfico de barras del nodo Grado Molienda.

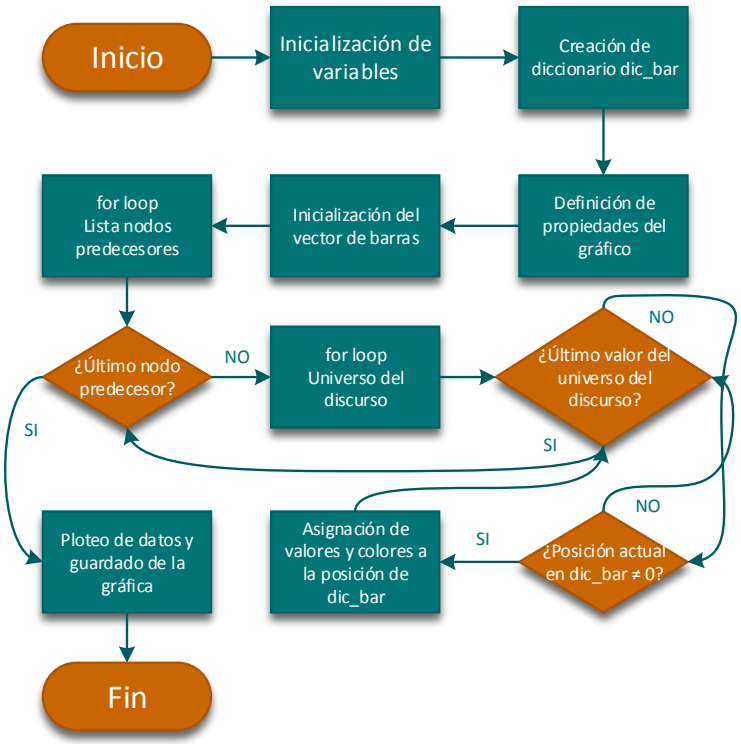

Figura 4.6: Diagrama de flujo del algoritmo de generación de gráficos de barras.

El diccionario *dic_bar* al que se hace referencia en el diagrama de flujo es un diccionario de la forma:

```
dic_bar = {1: [a1 a2 a3 ...], 2: [b1 b2 b3 ...], 3: [c1 c2 c3 ...],
           4: [d1 d2 d3 ...], ...}
```

donde tendremos tantas entradas como valores en el universo del discurso del sistema (5 en nuestro caso) y donde cada entrada tendrá tantos valores como nodos precedentes tenga el nodo actual, representando estos valores la altura que debe tener la barra que representará el impacto de ese nodo precedente sobre ese valor del universo del discurso del nodo sucesor.

La sintaxis empleada para invocar esta función, acción que se realiza en bucle al inicio del método *visualize_results_graph* y sólo si el parámetro *show_bars* está activado, es la siguiente:

```
gen_bars(name,
         dic_w,
         kernels,
         crisp_value,
         list_colores,
         folder,
         prop,
         markercolor)
```

donde:

- name: nombre de guardado de la imagen. En las llamadas a la función empleamos el nombre del nodo que representa.
- dic_w: diccionario que alberga la información de los nodos precedentes y el peso de los mismos sobre el nodo actual. Es la base para la creación del diccionario dic_bar comentado anteriormente y presenta la forma siguiente:

  ```
  dic_w = {Nodo 1: [e1 e2 e3 ...], Nodo 2: [f1 f2 f3 ...],
           Nodo 3: [g1 g2 g3 ...], ...}
  ```

  donde tendremos tantas entradas como nodos precedentes tenga el nodo y donde cada entrada tendrá tantos valores como tenga el universo del discurso.
- kernels: vector con el valor del intervalo del universo del discurso del sistema.

- crisp_value: valor nítido del nodo a representar.
- list_colores: lista con los colores que se van a emplear en la representación, previamente asignados a cada nodo.
- folder: directorio donde guardaremos las imágenes de los nodos.
- prop: parámetro necesario para determinar el tamaño de los graficos.
- markercolor: color del puntero del valor nítido y de su valor numérico.

Para conseguir un mayor orden en el directorio donde se guardan las imágenes de los nodos se ha desarrollado un algoritmo que permite crear una carpeta dentro de la cuál guardaremos dichas imágenes. Además, el algoritmo está preparado para detectar si una carpeta con el nombre deseado ya se encuentra en el directorio, de forma que añade un número identificador a la nueva carpeta, de la forma: *carpeta_ejemplo, carpeta_ejemplo (1), carpeta_ejemplo (2)...*

A continuación podemos observar tanto el diagrama de flujo (figura 4.7) de la función como la sintaxis de la misma:

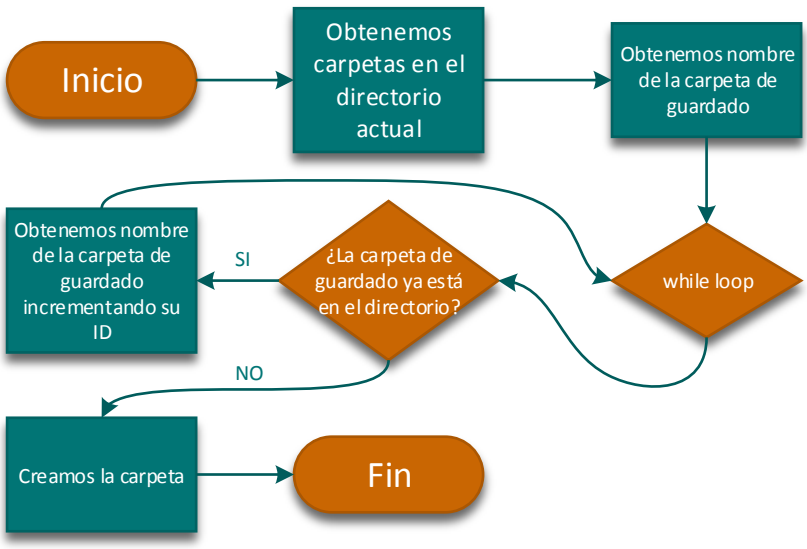

Figura 4.7: Diagrama de flujo del algoritmo de generación de carpetas contenedoras.

```
    _create_image_folder(self,
                         path,
                         base = '_images_folder'):
```

donde:

- self: engloba todo lo referente a la clase que estemos empleando, *FuzzyCognitiveMap* en este caso, pudiendo llamar a cualquier atributo o método de la misma.
- path: primera parte del nombre que daremos a la carpeta contenedora.
- base: segunda parte del nombre que daremos a la carpeta contenedora. Como valor predeterminado se empleará *'_images_folder'*.

4.1.3. Modificaciones adicionales

Como modificaciones visuales adicionales al bloque de representación podemos destacar dos fundamentalmente:

- **Color de la fuente de los nodos**: dado que los colores de las paletas empleadas presentan equivalencias en escala de grises muy dispares, se hace necesario modificar el color de la fuente de los nodos para que su lectura sea sencilla y evitar de esta forma combinaciones de colores que nos dificulten la comprensión del título, como serían por ejemplo azul oscuro y negro o amarillo claro y blanco (tabla 4.5).

Color incorrecto	Color incorrecto
Color correcto	Color correcto

Tabla 4.5: Comparativa de colores de la fuente del nodo.

Dicha transformación se realiza mediante la llamada en bucle para cada nodo de la función *node_font_color*, cuyo código, por su brevedad y simpleza, mostramos completo a continuación:

```
node_font_color(color_str):

        red   = int(color_str[1:3], 16)
        green = int(color_str[3:5], 16)
        blue  = int(color_str[5:7], 16)

        gray_scale_value = '#000000' if (red * 0.2126 + green *
            0.7152 + blue * 0.0722) > 0.5 * 255  else  '#FFFFFF'

        return color_fin
```

Dado el color del nodo como entrada en formato hexadecimal obtenemos los valores RGB y aplicamos la ecuación (4.1)

$$gray\_scale\_value = 0{,}2126 \cdot R + 0{,}7152 \cdot G + 0{,}0722 \cdot B \tag{4.1}$$

Finalmente asignamos a la fuente del nodo color negro si este valor es mayor que 127.5 y blanco si es menor o igual a dicho valor, por ser el rango total de [0, 255].

Mediante esta técnica modificamos el color tanto del nombre de cada nodo como de los valores en ambos ejes cuando se representen los gráficos de barras. Como ejemplo se pueden comprobar las tablas 4.1, 4.2, 4.3 y 4.4.

- **Estilo de los arcos internodales**: los arcos se ven representados de distinta forma en función del peso que tenga su nodo precedente sobre el sucesor al que está unido. El grosor será un valor en el intervalo (0,5], proporcional al porcentaje de peso, asignandose el valor 5 a un 100 % de peso. En el caso de que el nodo no tenga ningún peso (0 %) sobre el nodo sucesor debido a su propio valor nítido su arco se representará con línea punteada y grosor 5.

En los siguientes ejemplos se muestran los resultados derivados de estas modificaciones, así como algunas de las características del método *visualize_results_graph* comentadas al inicio de este capítulo.

En el primer ejemplo (figura 4.8) podemos observar una simplificación del nodo *Emulsión Pasta Corregida* realizada con la paleta de colores otoñales. Las características

mencionadas anteriormente que pueden relacionarse con este ejemplo son las siguientes:

- La variación de grosor de los arcos internodales, destacando el peso relativo del nodo predecesor sobre el sucesor, como podemos ver claramente en la relación *Madurez - Emulsión Pasta*.
- Visualización del arco punteado cuando el nodo predecesor no tiene ningún peso sobre el sucesor, como podemos comprobar tanto en la relación *Desgaste Martillos - Emulsón Pasta*, como en la relación *Relación Pulpa-Hueso - Emulsón Pasta*.
- Empleo de *show_relation_matrices = True*, de forma que se muestra el resultado de las matrices de relación causal multiplicadas por las intensidades de las relaciones.
- Al introducir un vector con los nombres de los nodos considerados parámetros del proceso sustituimos su forma elíptica original por una nueva forma hexagonal.

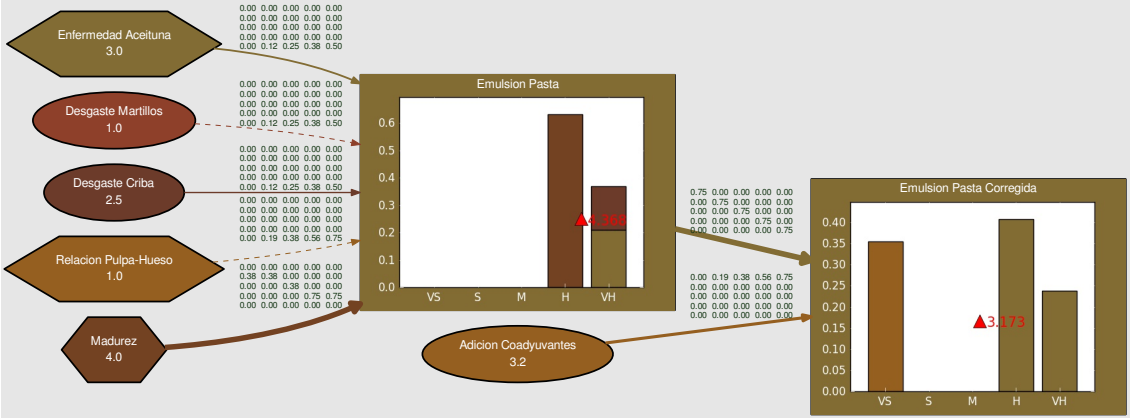

Figura 4.8: Ejemplo 1: simplificación del nodo *Emulsión Pasta Corregida*.

En el segundo ejemplo (figura 4.9), realizado como simplificación del nodo *Incremento Temperatura Molino*, podemos observar tres nuevas modificaciones:

- La variación del color de la fuente del nodo, blanca para colores más oscuros como el rojo y el azul, y negra para colores más claros como el amarillo.
- Empleo de *show_contribution_values = True*, de manera que podemos comprobar numéricamente el impacto sobre cada valor del intervalo del universo del discurso, a modo de añadido a la comprobación gráfica que suponen el tamaño de las barras.

- Empleo de *show_relation_matrices = 'Impact'*, de forma que se muestra el impacto porcentual que el nodo predecesor está teniendo sobre el sucesor. Dado que el impacto total sobre el nodo está ponderado sobre 1, se hace evidente que la suma del tamaño de las barras, o lo que es lo mismo, la suma de los impactos parciales de un nodo sucesor será igual al impacto porcentual mostrado de esta forma.

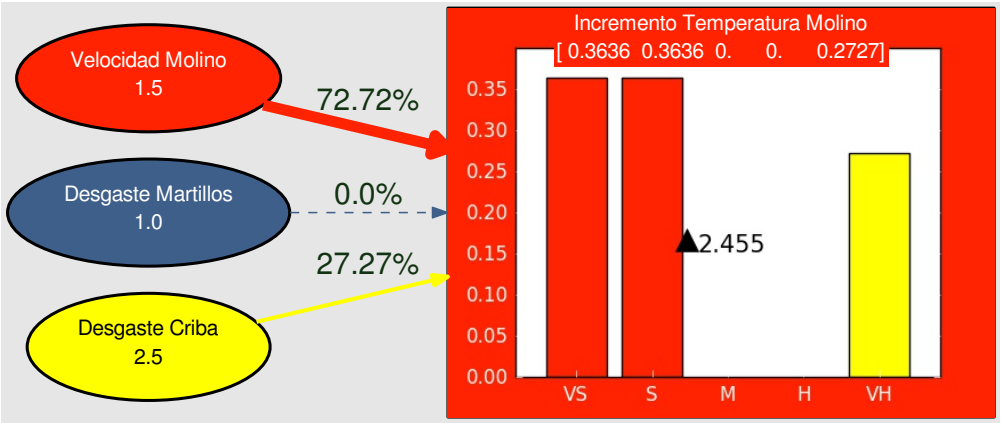

Figura 4.9: Ejemplo 2: simplificación del nodo *Incremento Temperatura Molino*.

Como último ejemplo, y una vez finalizada la presentación de todo el bloque de representación, mostramos a continuación un ejemplo del empleo del método *visualize_results_graph*, para un sistema completo, mostrando los impactos porcentuales de cada nodo sobre sus precedentes (figura 4.10). Puede consultarse en el anexo II el resultado final de los dos sistemas completos, mostrando las cuatro paletas distintas desarrolladas (figuras II.1, II.2, II.3 y II.4).

Por último, dado que ambos sistemas tienen nodos en común podemos realizar un modelo conjunto, teniendo en cuenta que esta unión de ambos sistemas no constituye uno de los sistemas del PEAOV. El objetivo de la realización de esta unión radica en que el lector pueda comprobar una posible combinación de la asignación aleatoria de colores, así como mostrar lo difícilmente interpretable que puede ser un sistema experto cuando la cantidad de nodos que contiene comienza a crecer, lo que hace necesario el sistema de utilidades desarrolladas y expuestas en el **bloque de simplificación**. Para este ejemplo emplearemos la selección aleatoria de colores (figura II.5).

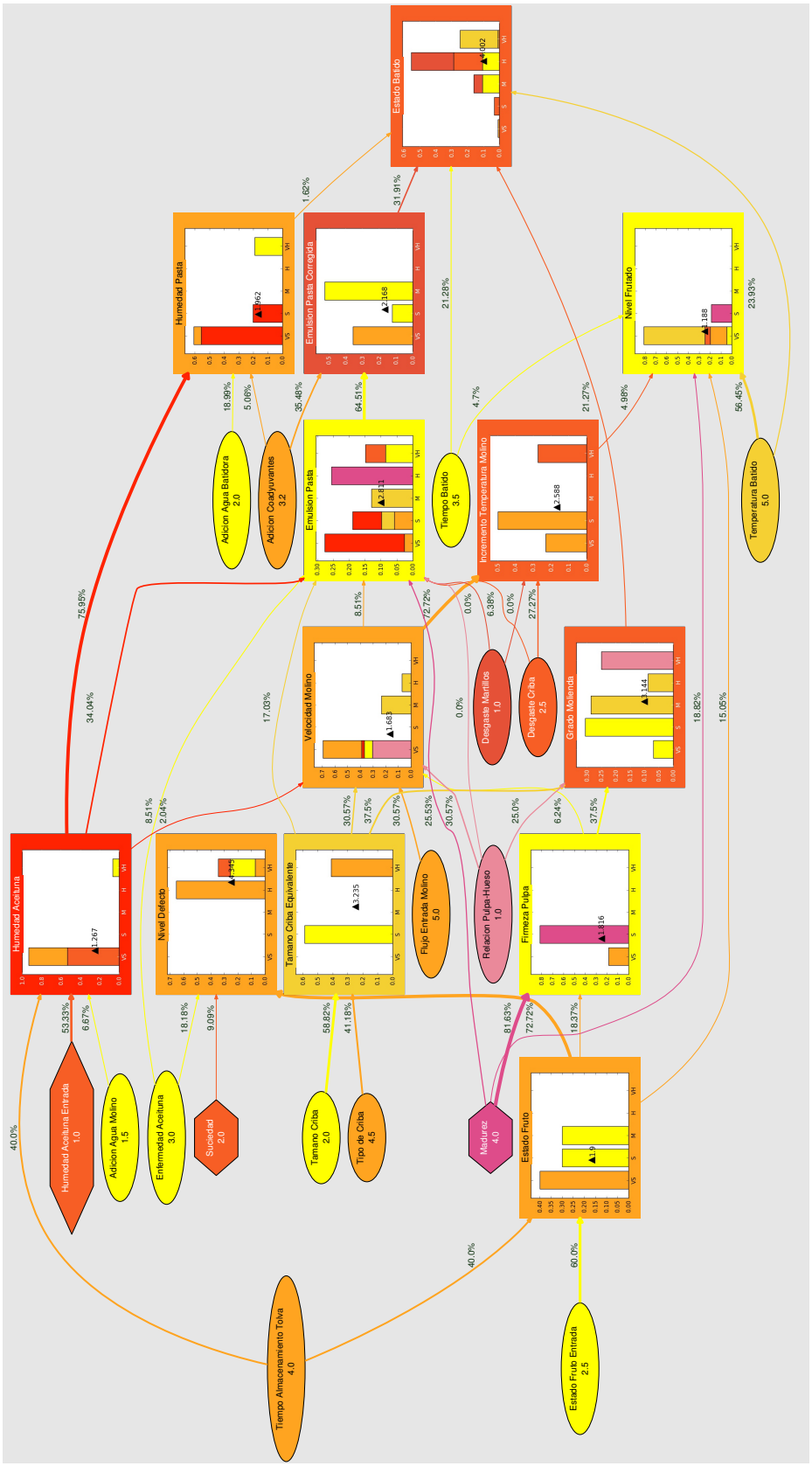

Figura 4.10: Representación gráfica de un sistema generado por el método *visualize_results_graph*.

4.2. Bloque de simplificación

4.2.1. Extracción de nodos y simplificación del sistema

Como ya se ha mencionado anteriormente y podiamos ver ejemplificado con la figura II.5, cuando un sistema experto está compuesto por una cantidad de nodos que supera el medio centenar aproximadamente su tratamiento gráfico y comprensión comienzan a dificultarse debido tanto a razones gráficas (resolución de la representación gráfica) como a razones cognitivas (a mayor número de nodos mayor número de relaciones entre ellos, lo que dificulta su comprensión).

Es por ello que la primera utilidad que vamos a desarrollar consiste en la *extracción* de nodos del sistema, esto es, seleccionar el nodo o los nodos que deseemos y representar sólo los nodos y arcos internodales que tengan relación con ellos. Pese a que lo habitual será querer conocer los nodos que tienen relación con los seleccionados aguas arriba (*backward*), la utilidad desarrollada también permite realizar tanto un análisis aguas abajo (*forward*), es decir, seleccionar un nodo y extraer los nodos sobre los que éste tiene incidencia, como un análisis completo, combinación de los dos anteriores.

Como se hace evidente al observar el problema con el que tratamos, la base de esta utilidad es un algoritmo recursivo denominado algoritmo de *recorrido de árbol*, muy empleado en ciencias de la computación para el recorrido de estructuras de datos jerárquicas.

En primera instancia, los algoritmos desarrollados para esta utilidad se concibieron con el objetivo de obtener **todos** los nodos relacionados con el nodo extraído, fuese cual fuese el tipo de análisis. Sin embargo, como mejora se decidió implementar una variante del algoritmo de recorrido de árbol, de forma que no se *visitasen* todos los nodos relacionados, sino sólo los nodos con más peso porcentual sobre el nodo sucesor, parámetro al que llamaremos **grado de influencia** G.

Los diagramas de flujo de los algoritmos desarrollados para esta utilidad son los correspondientes a las figuras 4.11 (extracción y representación, denominado *extract_node*) y 4.12 a 4.14 (recorrido de árbol, denominado *obtain_subgraph*):

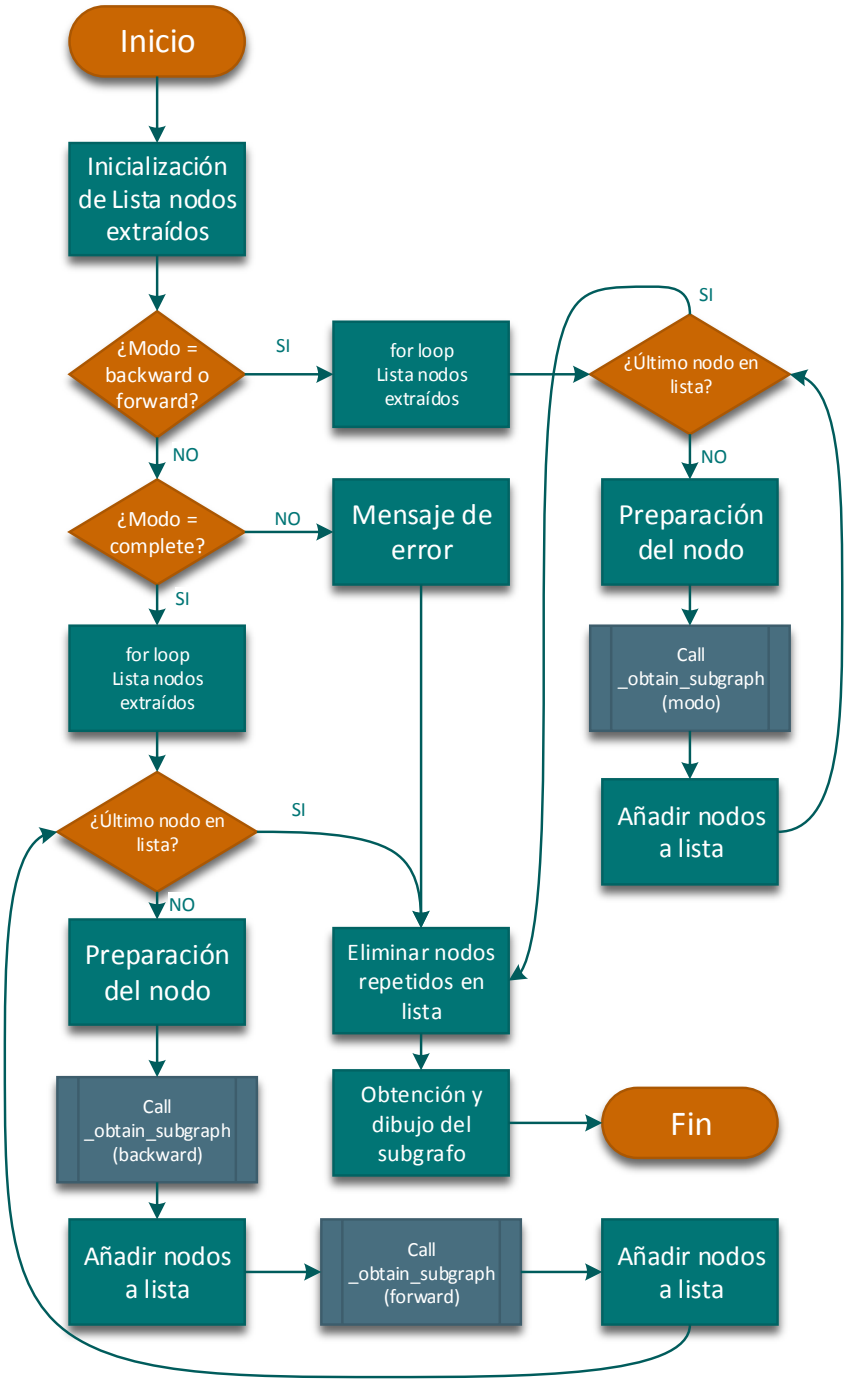

Figura 4.11: Diagrama de flujo del algoritmo de extracción y representación del subgrafo.

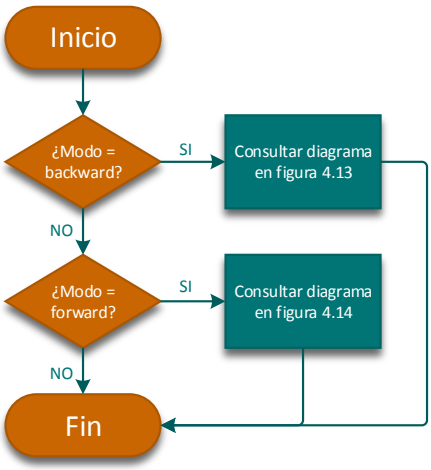

Figura 4.12: Diagrama de flujo del algoritmo de recorrido de árbol. Inicio.

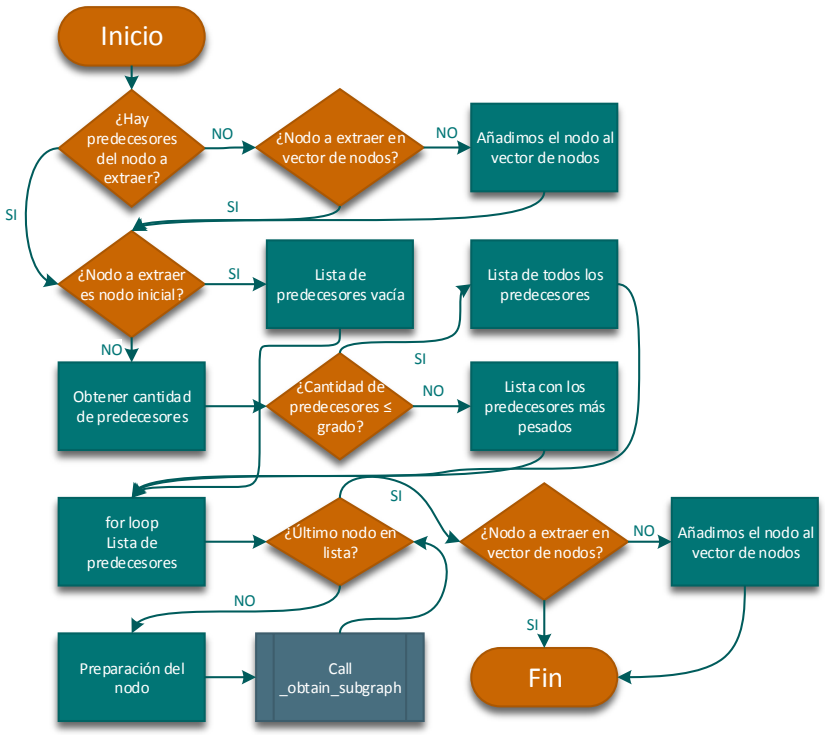

Figura 4.13: Diagrama de flujo del algoritmo de recorrido de árbol. Modo *backward*.

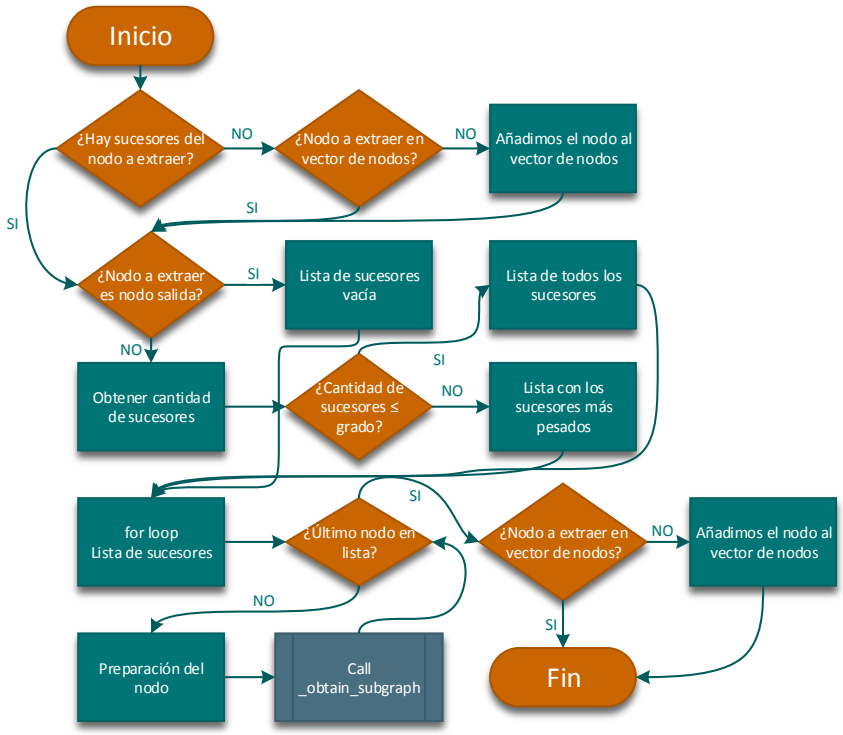

Figura 4.14: Diagrama de flujo del algoritmo de recorrido de árbol. Modo *forward*.

La sintaxis de dichos algoritmos es la siguiente:

```
extract_node(self,
             lista_nodos_extraidos,
             nombre_graf,
             grade = float('inf'),
             mode = 'backward',
             path = None)
```

donde:

- self: engloba todo lo referente a la clase que estemos empleando, *FuzzyCognitiveMap* en este caso, pudiendo llamar a cualquier atributo o método de la misma.
- lista_nodos_extraidos: lista que contiene al nodo o a los nodos que deseamos extraer.
- nombre_graf: nombre que daremos al archivo pdf con el grafo final.

- grade: grado de influencia a representar en la extracción. Se emplea infinito como valor predeterminado, de forma que si no se indica ningún valor se extraerán todos los nodos relacionados.
- mode: modo de análisis. Se emplea *backward* como valor predeterminado.
- path: ruta del directorio donde guardamos el archivo pdf. Predeterminadamente, si no se elige ninguna, el propio método asigna como directorio el mismo elegido en la creación de la clase.

```
    _obtain_subgraph(self,
                     nodo_extraido,
                     rest,
                     grade,
                     mode = 'backward')
```

donde:

- self: engloba todo lo referente a la clase que estemos empleando, *FuzzyCognitiveMap* en este caso, pudiendo llamar a cualquier atributo o método de la misma.
- nodo_extraido: nodo a extraer en cada iteración del bucle del algoritmo *extract_node*.
- rest: lista con los nodos extraídos hasta el momento en la función recursiva.
- grade: grado de influencia a representar en la extracción.
- mode: modo de análisis. Se emplea *backward* como valor predeterminado.

Para la muestra de ejemplos partiremos de la figura II.1. Mostraremos ejemplos de extracción realizando los tres tipos de análisis, divididos de la siguiente forma:

- Análisis *backward* desde los grados de influencia 1 a 5 del nodo final *Estado Batido*.
- Análisis *backward* de grado de influencia 1 de los grupos de nodos *Estado Batido* y *Nivel Frutado*; y *Humedad Aceituna* y *Nivel Frutado*.
- Análisis *forward* desde los grados de influencia 1 a 3 del nodo inicial *Madurez*.
- Análisis completo desde los grados de influencia 1 a 3 del nodo intermedio *Emulsion Pasta*.

En la figura 4.15 mostramos un ejemplo de extracción, sobre el que podremos analizar algunas de las características y peculiaridades de la utilidad desarrollada. La extracción se ha realizado sobre el nodo *Madurez*, aplicando un análisis *forward* de grado 2.

De entrada podemos comprobar como, pese a ser un análsis de grado 2, del nodo *Madurez* parten tres arcos, hacia los nodos *Firmeza Pulpa*, *Emulsion Pasta* y *Nivel Frutado*. Podemos cercionarnos de que los dos primeros son los que llevan un mayor peso del nodo en cuestión, sin embargo, dado que *Nivel Frutado* también aparece en el grafo debido al impacto recibido desde *Incremento Temperatura Molino*, el arco que en el sistema completo los une también se representa aquí. Además observamos que *Incremento Temperatura Molino* tiene un único sucesor, debido obviamente a que también es el único sucesor que tiene en el sistema completo.

Volviendo al inicio del grafo podemos observar como el nodo *Firmeza Pulpa* presenta dos barras provenientes de los nodos *Madurez* y, observando el modelo completo, *Estado Fruto*. El gráfico de barras de cada nodo siempre se mostrará completo, de forma que se pueda conocer la cantidad de precedentes totales que se tienen y sus pesos respecto a los que sí han sido representados, pese a que desconozcamos qué nodos son realmente, característica que nos es indiferente dado que no forman parte directa de la extracción.

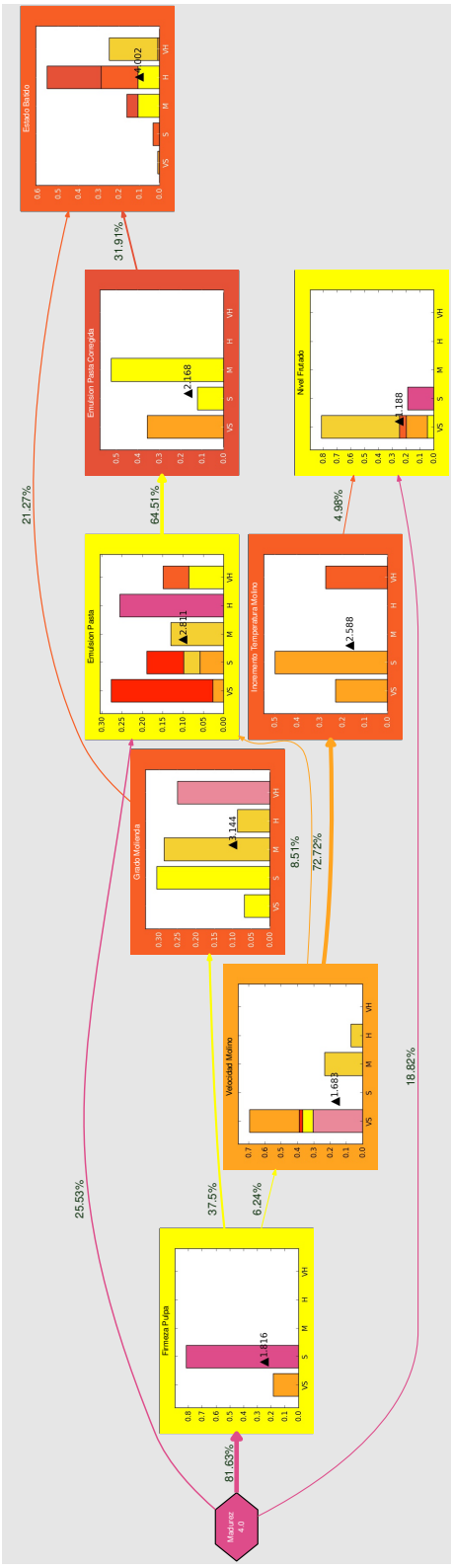

Figura 4.15: Análisis *forward* de grado 2 de *Madurez*.

Para una mejor observación de los grafos (figuras de III.1 a III.13) obtenidos estos se han representado en el anexo III, de forma que puedan representarse con una adecuada resolución.

Una vez consultados los grafos podemos extraer una serie de conclusiones derivadas del empleo de la utilidad a modo de generalidades:

- Pese a que sólo se representen los nodos correspondientes a un determinado grado de influecia *(G)*, las barras de impacto de cada nodo se representan en su totalidad (figuras III.1 y III.8). Esto permite tener conocimiento de que, pese a que sólo haya un determinado número de precedentes representados, puede haber un número distinto de nodos precedentes totales *(n)*. Este número total puede saberse a través de la cantidad de barras de colores diferentes representadas.

- No deben malinterpretarse los pesos relativos de las simplificaciones del sistema. Por ejemplo, en la figura III.1 el nodo *Humedad Aceituna Entrada* podría ser interpretado como el nodo inicial con mayor peso sobre el nodo *Estado Batido*, pero eso no es correcto. *Humedad Aceituna Entrada* es el nodo con mayor peso sobre *Humedad Aceituna*, éste a su vez lo es sobre *Emulsión Pasta*, y así sucesivamente, pero el escenario en el que una extracción de grado 1 correspondiese con la situación antes descrita se correspondería con un grafo en el que cada nodo tiene un único nodo sucesor.

- Cuando el grado de influencia $G > 1$ se intentará representar hasta G nodos precedentes. Sin embargo, si el número total de nodos precedentes es $n < G$, sólo se representarán los n nodos precedentes del nodo extraído (figuras III.3 y III.9).

- Las relaciones internodales entre dos nodos que aparezcan en el grafo siempre se representarán. Podemos comprobar esta situación en las figuras III.5 y III.9, entre los nodos *Relacion Pulpa-Hueso*, *Grado Molienda* y *Emulsion Pasta*, y entre los nodos *Incremento Temperatura Molino*, *Madurez* y *Nivel Frutado*, respectivamente. Esto puede conllevar, como se aprecia en las figuras III.9 y III.10, a resultados iguales con grados de influencia distintos.

- Si bien se pueden dar como entrada dos o más nodos con precedentes comunes (figura III.6), ésta no es una obligación como puede comprobarse en la figura III.7.

- Todos las conclusiones expuestas son válidas para cualquiera de los tres tipos de análisis disponibles.

4.2.2. Extracción completa

La utilidad de extracción de nodos nos permite obtener el grafo simplificado de una lista de nodos determinada, tal y como vimos en la sección anterior. Sin embargo, en ocasiones será más conveniente obtener una **simplificación de todos los nodos simultáneamente**, es decir, realizar alguno de los tres tipos de análisis vistos anteriormente (*backward*, *forward* y completo).

Mediante el empleo de la utilidad de extracción completa, basada en el algoritmo denominado *extract_all*, podremos generar una nueva carpeta dentro del directorio actual de trabajo que albergará todo el conjunto de nodos simplificados. Como base de cálculo se emplea el algoritmo *extract_node*, explicado en la sección anterior.

El objetivo de la utilidad es meramente la demostración del funcionamiento y la potencialidad del algoritmo, vistas ambas como una extensión de la utilidad de extracción de nodos. Por este motivo se ha impuesto que la simplificación que se realiza imponga un **grado de influencia** infinito, aunque se pueden obtener grados de influencia diferentes con muy ligeras modificaciones en el código si se desease.

La sintaxis del algoritmo desarrollado se muestra a continuación, mientras que el diagrama de flujo del mismo se muestra en la figura 4.16.

```
    extract_all(self,
                mode = 'backward')
```

donde:

- self: engloba todo lo referente a la clase que estemos empleando, *FuzzyCognitiveMap* en este caso, pudiendo llamar a cualquier atributo o método de la misma.
- mode: modo de análisis. Se emplea *backward* como valor predeterminado.

Los nodos que se analizan son diferentes en función del tipo de análisis que se solicite:

- Análisis *backward*: se analizan los nodos intermedios y los nodos finales.
- Análisis *forward*: se analizan los nodos intermedios y los nodos iniciales.
- Análisis completo: se analizan todos los nodos del sistema. Cabe destacar que se obtendrán resultados repetidos respecto a los anteriores análisis: el análisis completo de un nodo final es igual a sus análisis *backward*, mientras que el análisis completo de un nodo inicial es igual a su análisis *forward*.

El nombre de las carpetas creadas siempre tendrá la siguiente forma:

$$Nombre\ del\ modelo\ +\ tipo\ de\ analisis\ +\ extraction$$

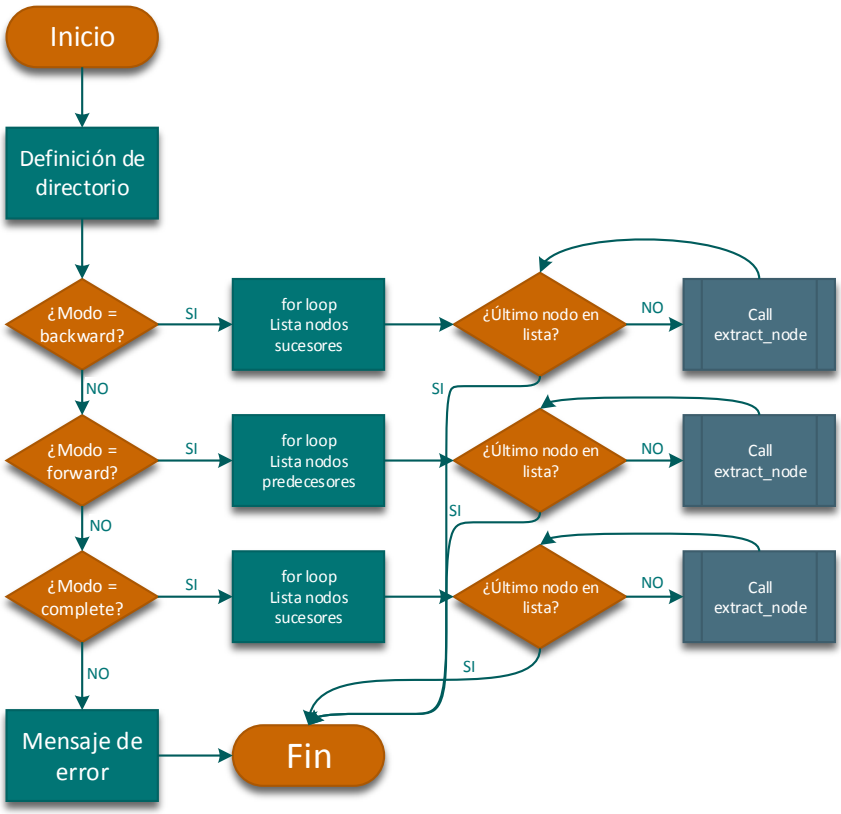

Figura 4.16: Diagrama de flujo del algoritmo de extracción completa

4.2.3. Parametrización de nodos

Una de las líneas de investigación más importantes dentro del estudio de los sistemas expertos basados en mapas borrosos cognitivos es la de la **optimización** de los sistemas, es decir, cuales son las variables de decisión óptimas, con unos parámetros del proceso fijados, que nos permiten obtener unas variables de salida objetivo. Esta situación, resumida aquí en gran medida, es aún dificil de resolver, empleándose por el momento métodos heurísticos o algoritmos de búsqueda de fuerza bruta.

Como se hace patente, la dificultad del problema expuesto excede en demasía el nivel requerido para el presente proyecto, por lo tanto se ha decidido elaborar una utilidad que pueda emplearsele como sustituto simple.

El objetivo de la utilidad de parametrización de nodos es la obtención de curvas que nos permitan conocer la evolución de una variable de salida en función de una serie de variables de entrada, que pueden ser tanto variables de decisión como parámetros del proceso. La idea es saber qué valor tendría el nodo de salida ante variaciones únicamente de un nodo de entrada concreto.

La sintaxis del algoritmo desarrollado y su diagrama de flujo (figura 4.17) pueden consultarse a continuación:

```
parametric(self,
           lista_entrada,
           lista_salida,
           datos_iniciales,
           dic_res,
           paso):
```

donde:

- self: engloba todo lo referente a la clase que estemos empleando, *FuzzyCognitiveMap* en este caso, pudiendo llamar a cualquier atributo o método de la misma.
- lista_entrada: lista que contiene los nodos que quieren emplearse como entradas (variables independientes). Si alguno de sus elementos no es un nodo inicial del sistema, se elimina de la lista.

- lista_salida: lista que contiene los nodos que quieren emplearse como salidas (variables dependientes). Si alguno de sus elementos es un nodo inicial del sistema, se elimina de la lista.
- datos_iniciales: diccionario que incluye los valores iniciales aplicados a los nodos iniciales (tabla 3.1).
- dic_res: diccionario que incluye los resultados iniciales del modelo.
- paso: variación en los valores de las variables de entrada. El intervalo de cálculo será el determinado por los valores extremos del universo del discurso.

Si bien se puede emplear el paso que se desee, siempre que sea divisor real de la diferenecia entre los valores extremos del universo del discurso, no es lógico, para el caso práctico que nos ocupa, emplear un valor distinto a 1, dado que esa es la diferencia entre los distintos valores del universo del discurso. Valores mayores darían poca información y valores menores pueden generar tiempos de procesado demasiado elevados.

Para los ejemplo mostrados a continuación se ha empleado un paso de 0.01, de forma que las gráficas sean más precisas y presentan una estética mejor, aunque se incumpla lo dicho en el párrafo previo. Los cuatro ejemplos mostrados presentan los mismos nodos independientes: *Estado Fruto Entrada*, *Tiempo Batido* y *Relacion Pulpa-Hueso*, mientras que los nodos dependientes elegidos han sido *Estado Batido*, *Nivel Defecto*, *Nivel Frutado* y *Velocidad Molino*.

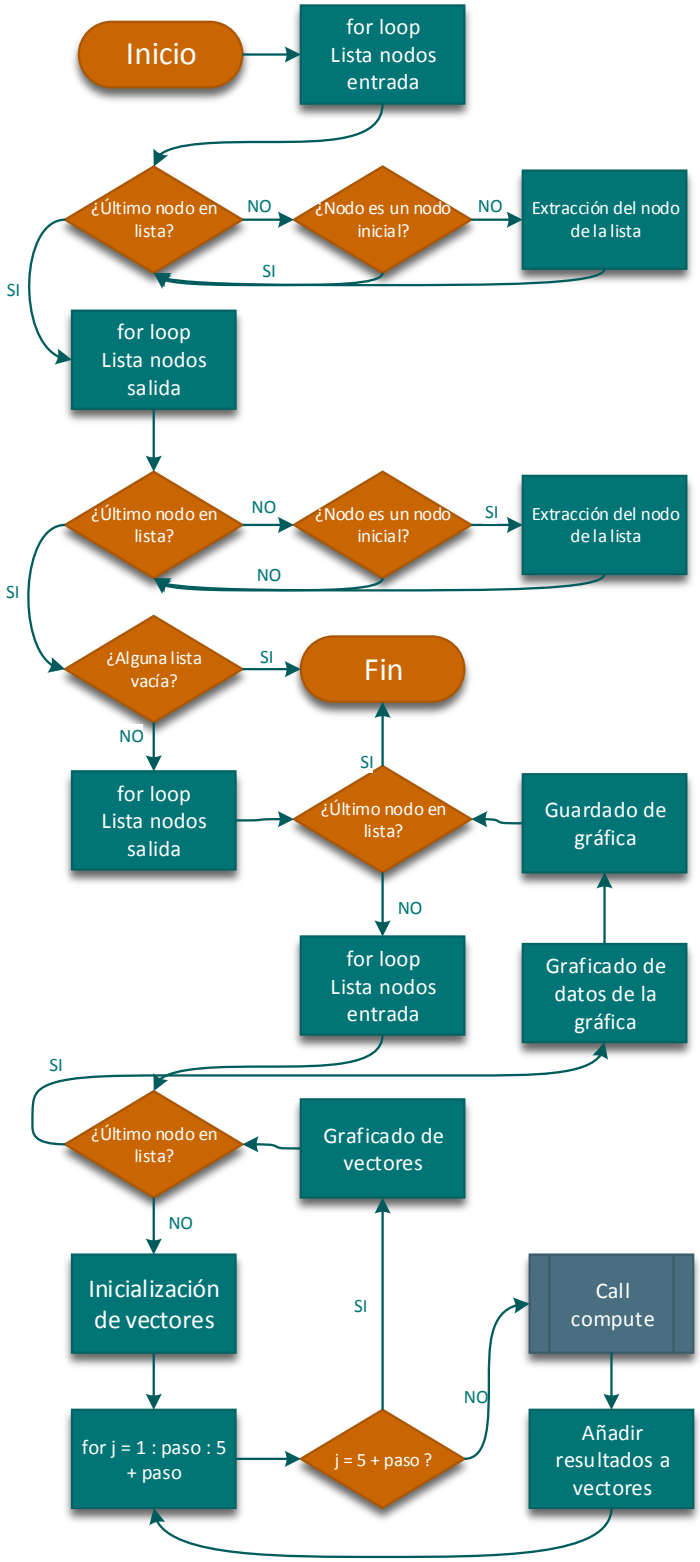

Figura 4.17: Diagrama de flujo del algoritmo de parametrización de nodos.

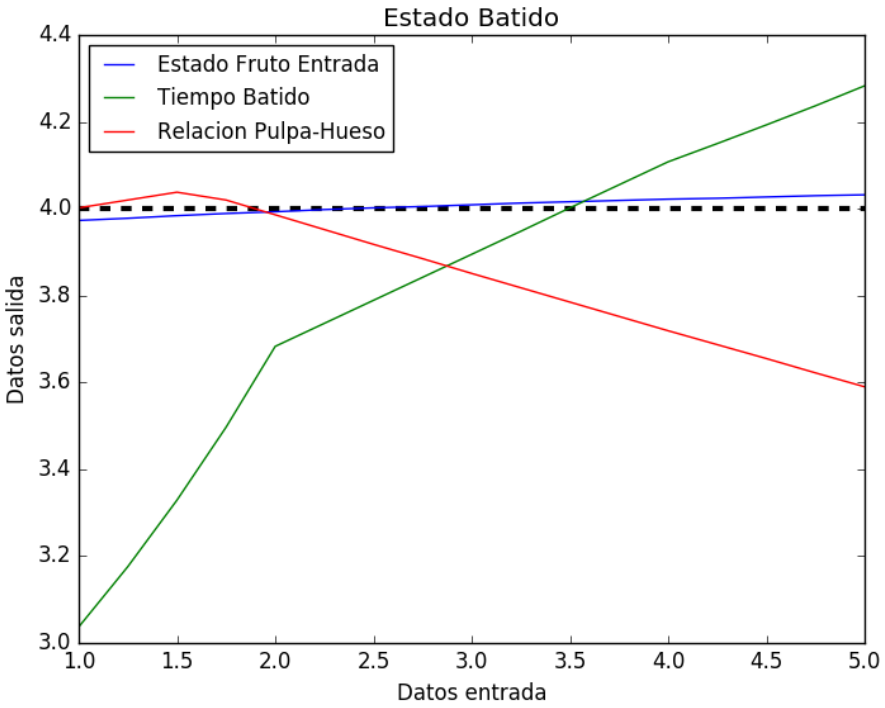

Figura 4.18: Ejemplo 1: parametrización de Estado Batido.

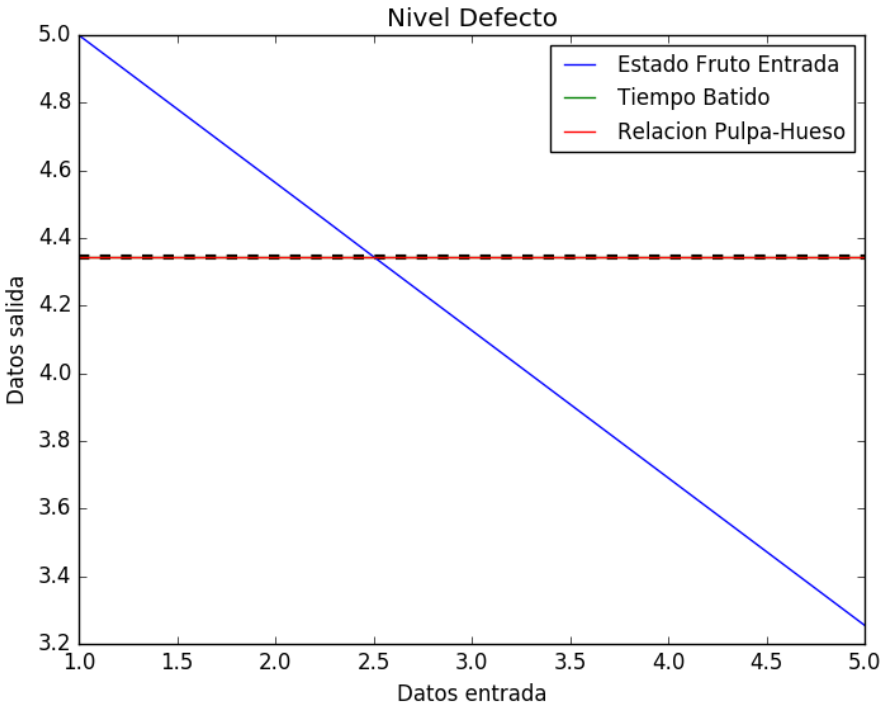

Figura 4.19: Ejemplo 2: parametrización de Nivel Defecto.

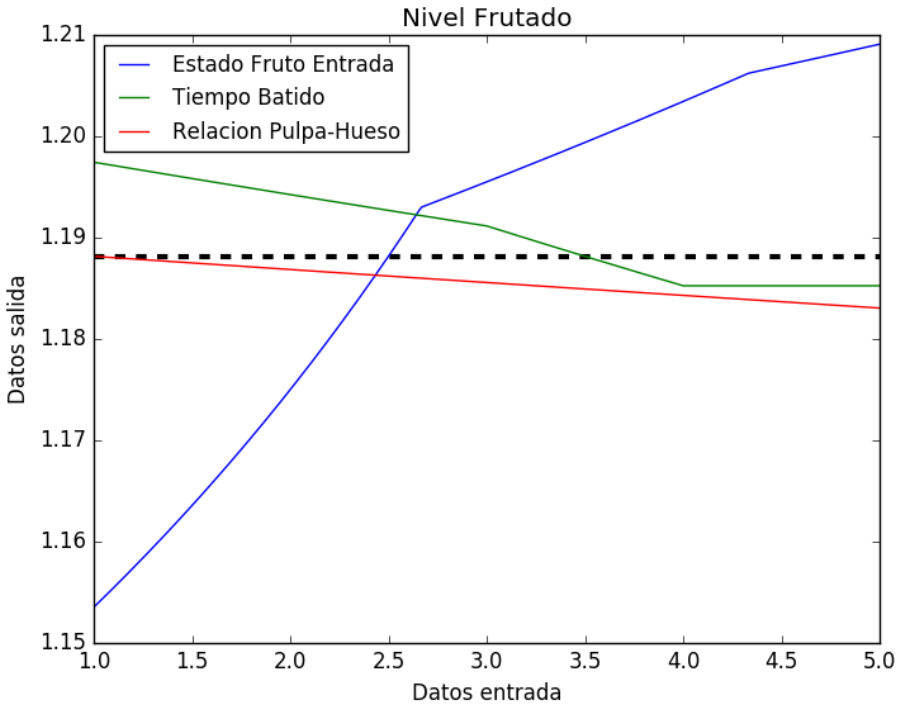

Figura 4.20: Ejemplo 3: parametrización de Nivel Frutado.

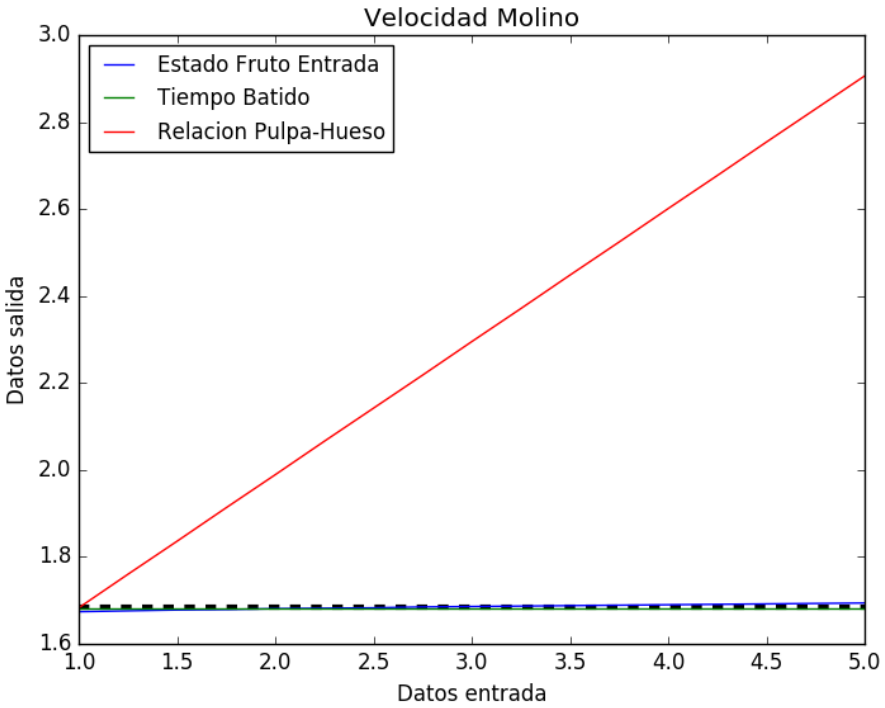

Figura 4.21: Ejemplo 4: parametrización de Velocidad Molino.

Hay que tener en cuenta una serie de características de los gráficos obtenidos, así como conclusiones a obtener a partir de los mismos:

- El intervalo del eje de abscisas depende del universo del discurso, mientras que el del eje de ordenadas depende de los valores mínimo y máximo de todo el conjunto de valores que puede tener la variable de salida.
- La leyenda siempre se coloca en el lugar donde menos estorbe a la visualización de las curvas.
- La línea negra punteada representa el valor original de la variable de salida, por lo tanto, los puntos de corte con las curvas representan los valores originales de las variables de entrada. Esta característica se puede comprobar mediante la tabla 3.1.
- Curvas ascendentes puras representan una relación bivalente positiva (*Tiempo Batido* en figura 4.18), mientras que curvas descendentes puras representan relaciones bivalentes negativas (*Estado Fruto Entrada* en figura 4.19).
- Curvas con tramos constantes representan una relación univalente (*Tiempo Batido* y *Relacion Pulpa-Hueso* en figura 4.20), mientras que líneas sobre la línea negra punteada indican que no hay ninguna relación entre ambos nodos (*Velocidad Molino* y *Relacion Pulpa-Hueso* en figura 4.21), situación que puede corroborarse con una extración *backward* del nodo de salida con grado de influencia infinito o con una extracción *forward* del nodo inicial con el mismo grado de influencia.

4.2.4. Cálculo de peso total internodal

Podríamos definir el peso total internodal como el peso relativo de cualquier nodo sobre cualquier otro del sistema, sin necesidad de que sean dos nodos contiguos (pareja *predecesor* - *sucesor*). Esta utilidad, a la que nombramos como *get_total_weight*, se presenta de gran utilidad en los casos en los que se presenta un sistema con una gran cantidad de nodos y, sobre todo, una gran cantidad de relaciones internodales, de forma que se hace virtualmente imposible discernir el peso real que se está ejerciendo sobre un nodo final desde los nodos iniciales o intermedios no precedentes directos.

La utilidad está basada en el empleo de una función ya integrada en NetworkX, denominada *all_simple_paths*, que permite obtener, a partir de un nodo dado como origen y otro como objetivo, todos los caminos posibles a lo largo del grafo entre esos dos nodos. Una vez obtenidos todos los caminos posibles simplemente realizamos el sumatorio del peso total sobre cada camino, como podemos observar a continuación entre los nodos x e y:

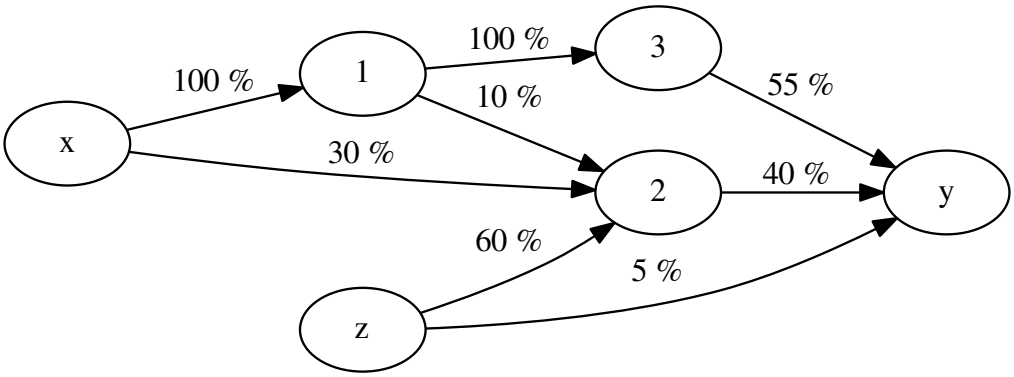

Figura 4.22: Ejemplo de grafo para cálculo de pesos totales.

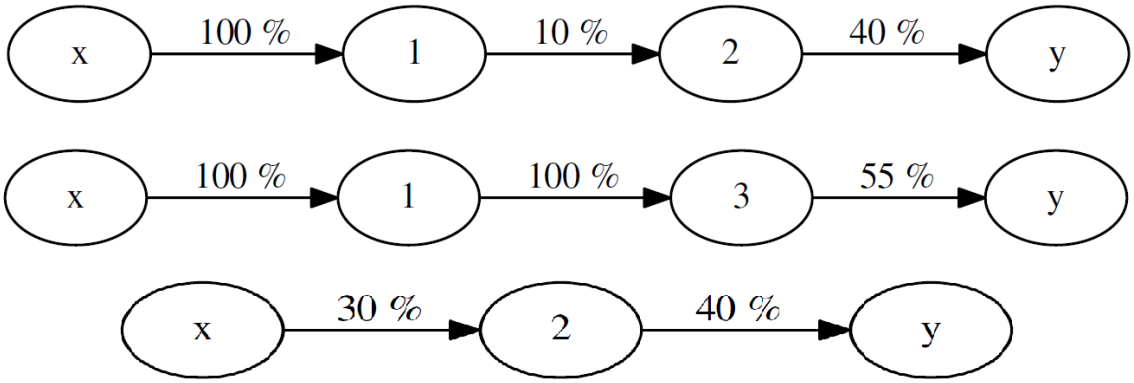

Figura 4.23: Caminos posibles entre los nodos x e y.

Vistos los caminos posibles entre los nodos x e y y los pesos relativos de todas las relaciones del grafo podemos calcular el peso total entre ambos nodos como sigue:

$$\hat{W}_{path_1} = 1 \cdot 0{,}1 \cdot 0{,}4 = 0{,}04 \tag{4.2}$$

$$\hat{W}_{path_2} = 1 \cdot 1 \cdot 0{,}55 = 0{,}55 \tag{4.3}$$

$$\hat{W}_{path_3} = 0{,}3 \cdot 0{,}4 = 0{,}12 \tag{4.4}$$

$$\hat{W}_t = \sum_k \hat{W}_{path_k} = 0{,}71 \equiv 71\,\% \tag{4.5}$$

La sintaxis de la función es la siguiente:

```
get_total_weight(self,
                 init,
                 end)
```

donde:

- self: engloba todo lo referente a la clase que estemos empleando, *FuzzyCognitiveMap* en este caso, pudiendo llamar a cualquier atributo o método de la misma.
- init: nodo inicial que actúa como origen del camino.
- end: nodo final que actúa como objetivo del camino.

Para los nodos iniciales del sistema de estudio y sus nodos finales se obtienen los siguientes resultados (tabla 4.6):

	Estado Batido	**Nivel Defecto**	**Nivel Frutado**
Adición Agua Batidora	0.31 %	0 %	0 %
Adición Agua Molino	0.55 %	0 %	0 %
Adición Coadyuvantes	11.4 %	0 %	0 %
Desgaste Criba	1.31 %	0 %	1.36 %
Desgaste Martillos	0 %	0 %	0 %
Enfermedad Aceituna	1.75 %	18.18 %	0 %
Estado Fruto Entrada	0.89 %	43.63 %	9.05 %
Flujo Entrada Molino	0.21 %	0 %	1.11 %
Humedad Aceituna Entrada	4.41 %	0 %	0.04 %
Madurez	11.86 %	0 %	19 %
Relación Pulpa-Hueso	5.85 %	0 %	1.11 %
Suciedad	0 %	9.09 %	0 %
Tamaño Criba	7.07 %	0 %	0.65 %
Temperatura Batido	23.93 %	0 %	56.45 %
Tiempo Almacenamiento Tolva	3.9 %	29.09 %	6.07 %
Tiempo Batido	21.28 %	0 %	4.7 %
Tipo Criba	4.95 %	0 %	0.46 %

Tabla 4.6: Pesos totales internodales para todas las combinaciones *Nodo Inicial - Nodo final* del caso de estudio.

Como podemos apreciar, muchos de los valores son de 0 %. Este resultado se puede deber a dos motivos únicamente:

1. El nodo no tiene ningún impacto sobre ninguno de sus sucesores que sí lo tengan sobre el nodo final en cuestión. El máximo exponente de este motivo lo encontramos en el nodo *Desgaste Martillos*, que al tener un valor nítido de 1, es decir, un desgaste muy reducido, no tiene influencia sobre sus nodos sucesores, dado que un nivel elevado de desgaste influirá negativamente en los mismos, mientras que un nivel reducido no mejorará el resultado, sino que no se presentará ningún efecto.

2. El nodo no es precedente del nodo objetivo en cuestión, como ocurre por ejemplo para la combinación *Tiempo Batido - Nivel Defecto* (figura II.1).

4.3. Resumen de utilidades

Una vez presentadas todas las utilidades y discutidas todas sus ventajas y potencialidades vamos a presentar un cuadro resumen con el nombre del método desarrollado y un ejemplo de los presentados durante este trabajo.

Utilidad	**Método**	**Ejemplo**
Visualización completa	*visualize_results_graph*	visualize_results_graph(dic_res, 'test_warm', list_nodos_fijos = list_nodos_fijos, show_bars = True, palette = 'warm', show_relation_matrices = 'Impact')
Extracción de nodos	*extract_node*	extract_node(['Estado Batido'], grade = 4, nombre_graf = 'estado batido grade 4, back')
Extracción completa	*extract_all*	extract_all('complete')
Parametrización de nodos	*parametric*	parametric(['Estado Fruto Entrada','Tiempo Batido', 'Relacion Pulpa-Hueso'],['Estado Batido'], dic_valores_iniciales, dic_res, 1)
Cálculo de pesos internodales	*get_total_weight*	get_total_weight('Estado Fruto Entrada', 'Estado Batido')

Tabla 4.7: Resumen de las utilidades desarrolladas.

Capítulo 5

CONCLUSIONES

La técnica de los Mapas Borrosos Cognitivos ha sido ampliamente validada a lo largo de toda la literatura técnica como una técnica muy útil para modelar y analizar sistemas complejos dinámicos [PS13].Podemos leer a continuación las conclusiones más importantes que pueden extraerse tras la realización del presente Trabajo Fin de Máster

Como se mencionó durante la justificación de este trabajo, la interpretabilidad de los sistemas expertos basados en lógica difusa, y más especialmente en mapas borrosos cognitivos, es una cualidad que representa un gran reto a conseguir. Mediante las modificaciones realizadas al código original de la tesis que nos ha servido como base y las utilidades desarrolladas, se ha logrado un aumento considerable de interpretabilidad de los mapas borrosos cognitivos.

Podemos hacer mención especial también a las conclusiones extraídas de la modificación del método de cálculo de los valores nítidos de los nodos del sistema, especialmente a la conclusión de que el impacto porcentual de un nodo no depende de su valor nítido en los casos en los que se ve relacionado con su sucesor mediante una relación de tipo bivalente.

Cualquier usuario no experto podría extraer datos con suma facilidad, simplemente empleando la utilidad de extracción de nodos, apoyada considerablemente en las mejoras visuales implementadas. Gracias al empleo de los gráficos de barras el usuario puede

discernir rápidamente el nivel de influencia presente entre dos variables, así como el peso relativo que tiene cada nodo predecesor sobre su sucesor, medida comprobable así mismo a través tanto del grosor de las flechas de representación de los arcos internodales como de la etiqueta de dichos arcos en caso de haber sido representada.

Las utilidades de parametrización de nodos y de obtención de pesos totales internodales hacen las veces de apoyo de la utilidad de extracción, pudiendo realizar barridos de datos y una mejora en la interpretabilidad de los pesos internodales, respectivamente. Incluso sin el empleo de utilidades de optimización, es posible mejorar el resultado final de un sistema simple mediante un simple tanteo gracias al gran aumento de interpretabilidad conseguido, el cual es el principal objetivo del presente Trabajo Fin de Máster.

Podemos concluir finalmente que el trabajo ha logrado su objetivo fundamental, basado en la correcta representación gráfica de los nodos y los impactos generados entre ellos, así como los objetivos secundarios establecidos, entendiendo por estos el desarrollo de las utilidades que tanto los tutores como el autor considerasen convenientes.

Capítulo 6

TRABAJO FUTURO

Las principales lineas de trabajo que podrían continuar con el realizado aquí podrían ser:

- **Combinación de paletas** de colores al combinar diferentes subsistemas de un sistema experto a los que ya hayamos asignado una paleta, en lugar de crear un único subsistema mayor.
- **Parametrización** de todas las variables estéticas del código (tamaño de fuente, posición de etiquetas...), para su modificación en función de los nodos del grafo representado.
- **Otras** líneas que permitiesen una mejora y optimización de los algoritmos desarrollados, de forma que se mejore el tiempo de computación de los mismos.

Una línea de trabajo harto interesante y planteada durante la realización de este proyecto consiste en la creación de **modelos interactivos**, de forma que se pueda manipular los valores de las variables y conseguir una modificación de los resultados finales instantáneamente, sin necesidad de ejecutar los modelos nuevamente para cada modificación.

Apéndice I

Documentación del código empleado

Se aporta a continuación la debida documentación tanto del código empleado como base concerniente a las utilidades desarrolladas, como de las propias utilidades desarrolladas.

En primer lugar mostraremos la documentación referente a las clases *FuzzyCognitiveMap*, *Nodo* y *Relation*. Dicha documentación ha sido generada mediante la función pydoc, implementada de serie en Python. En la segunda parte del apéndice se mostrarán los diagramas de llamadas que realizan las distintas utilidades desarrolladas. Dichos diagramas se han desarrollado mediante una librería externa denominada pycallgraph. Dado que se consideran importantes, también se representan las llamadas realizadas por los métodos *gen_graph* y *visualize_results_graph*. Las distintas imágenes se muestran según el orden de aparición dentro del presente trabajo.

I.1. Documentación de clases

FuzzyCognitiveMap

index
/home/alvaro/Escritorio/TFM_WD/Python/Enviado_Pablo/WD/TFM/FuzzyCognitiveMap.py

Created by Pablo Cano Marchal <pcano@ujaen.es>
11 Mayo 2016

Modified by Álvaro Garzón Casado <garzon@ujaen.es>
29 Agosto 2017

Modules

PIL.Image	logtools	os	subprocess
copy	numpy	pandas	sys
itertools	networkx	matplotlib.pylab	yaml

Classes

__builtin__.object

- FuzzyCognitiveMap
- Nodo
- Relation

class **FuzzyCognitiveMap**(__builtin__.object)

Class implementing a Fuzzy Cognitive Map.

Methods defined here:

__init__(self, relations_list, name='sistema', output_path='./', verbose=False, extra_string='', kernels=[1, 2, 3, 4, 5])

compute(self, dic_valores, do_checks=True)
Computes the value of the nodes of the system
dic_valores is a dictionary with the values of the nodes
that have fixed values

extract_all(self, mode='backward')
Extraction of complete subgraphs with maximum degree of all the nodes of the system.
A directory is generated and all subgraphs are stored inside.

Inputs:

- mode: mode of analysis. Backward is used as default value. Forward and complete are the other possible options.
  - Backward: backward propagation of the node. It will be run for intermediate and output nodes.
  - Forward: forward propagation of the node. It will be run for input and intermediate nodes.
  - Complete: backward and forward propagations of the node. It will be run for every node in the system.

extract_node(self, lista_nodos_extraidos, nombre_graf, grade=inf, mode='backward', path=None)
Extraction of subgraphs from a list of nodes given as input, being possible to choose a degree of influence and mode.

Inputs:

- lista_nodos_extraidos: list that contains the node or nodes that we want to extract.

- nombre_graf: name we will give to odf file with final graph.

- grade: degree of influence to represent in the extraction.
  Infinity is used as default value, so that if novalue is given,
  all related nodes will be extracted.

- mode: mode of analysis. Backward is used as default value.

- path: path of the directory where we save the pdf file.
  Predetermined, if none is chosen, the method itself assigns as
  the directory the same chosen in the creation of the class.

Outputs:

- rest: nodes set that compound the extraction subgraph.

gen_graph(self, path=None, show_relation_matrices=False, show_node_values=False, show_contribution_values=False, recompute_node_locations=True, orden_grafo='LR', fontsize=9, func_texto_nodo=<function <lambda>>)
Makes a graph that represents the nodes contained in the system and their relations.

get_total_weight(self, init, end)
Allows calculus of total weight of a single nodo over another one, not necessary a successor one.

Inputs:

- init: initial node that acts as path source.

- end: final node that acts as path target.

Outputs:

- total_weight: total weight between nodes.

parametric(self, lista_entrada, lista_salida, datos_iniciales, dic_res, paso)
```
Function that allows a graphical representation of the
relation among a list of input nodes and a list of output nodes.


Inputs:

- lista_entrada: list containing nodes that we can use as inputs (independent variables).
  If any of them isn't an initial node of the system, it's removed.

- lista_salida: list containing nodes that we can use as outputs (dependent variables).
  If any of them is an initial node of the system, it's removed.

- datos_iniciales: dictionary including default values applied to initial nodes.

- dic_res: dictionary including model initial results.

- paso: step between two consecutives values of the output variables.
  Calculus range is set by minimum and maximum values of the universe of discourse.
```

study_model_output(self, study_variables, fixed_value_nodes={}, default_value=3, num_partition_points=5)
```
Computes the ouput of the system for each
point in the cartesian product of study_variables
```

values(self, crisp=True)
```
Returns the current values of all the nodes in a dictionary
if crisp is True, returns crips values
if crisp is false, returns fuzzy_membership_vector
```

visualize_results_graph(self, dic_values, nombre_graf, list_nodos_fijos=[], dic_costos={}, path=None, show_relation_matrices=False, show_node_values=True, show_cost=False, show_contribution_values=False, show_bars=False, recompute_node_locations=True, palette='rand', orden_grafo='LR', fontsize=9, func_texto_nodo=<function <lambda>>, rewrite=False)
```
Defines a graph based on the system and plots it,
saving it in a file with path defined by self.output_path
```

Data descriptors defined here:

__dict__
```
dictionary for instance variables (if defined)
```

__weakref__
```
list of weak references to the object (if defined)
```

class **Nodo**(__builtin__.object)
```
This class is a data structure to hold the values associated with a node.
```

Methods defined here:

__eq__(self, other)
```
Also required for appropriate dict keys handling.
```

__hash__(self)
```
Override the __hash__ function
so that we can use it
as dict keys with different
instances.
```

__init__(self, variable)
```
We create the fields of the class.
```

__repr__(self)
```
Definition of the value shown as representative of the object.
```

__str__(self)
```
Definition of the value shown as representative of the object.
```

compute_value(*args, **kwargs)
```
Computes the value of the node for the given value of xtotal.
```

load_value(self, value)
```
Loads the specified value into the node.
```

populate_ones_and_kernels(self)
```
Initializes arrays and matrices.
```

Data descriptors defined here:

__dict__
```
dictionary for instance variables (if defined)
```

__weakref__
```
list of weak references to the object (if defined)
```

class **Relation**(__builtin__.object)
```
This class implements a relation as defined in the paper.

predecessor and successor are Variable objects
omega is a float.
R is a string containing the relation matrix
```

R is a string containing the relation matrix.

Methods defined here:

__init__(self, predecessor, successor, R, omega, dic_mapping_valores={'+': 1, '++': 1, '0': 0, '02': 0.2, '04': 0.4, '05': 0.5, '06': 0.6, '075': 0.75, '08': 0.8, '1': 1, ...})

__str__(self)
Definition of the value shown as representative of the object.

get_sparse_representation(self)
Returns the sparse representation of the matrix
(row_index, column_index, value).

Data descriptors defined here:

__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

Functions

fs = fsum(...)
fsum(iterable)

Return an accurate floating point sum of values in the iterable.
Assumes IEEE-754 floating point arithmetic.

iterator_cartesian_product(lista_variables, numero_puntos=5)
Gets a list of variables and a number of points
to partition the universe and returns a list with
the cartesian product of values for each variable
together with the variable.

iterator_diccionario_escenario(diccionario_base, variables_barridas, numero_puntos_en_universo=3)
Takes a base dictionary and returns an iterator that
gives a dictionary with
the values of each scenario
contained in the list.

Data

DEFAULT_DIC_MAPPING_VALORES = {'+': 1, '++': 1, '0': 0, '02': 0.2, '04': 0.4, '05': 0.5, '06': 0.6, '075': 0.75, '08': 0.8, '1': 1, ...}
FUENTE = 'Helvetica'
GROSOR_LINEA_NODO = 2
division = _Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192)
logger = <logging.Logger object>

I.2. Diagramas de llamadas

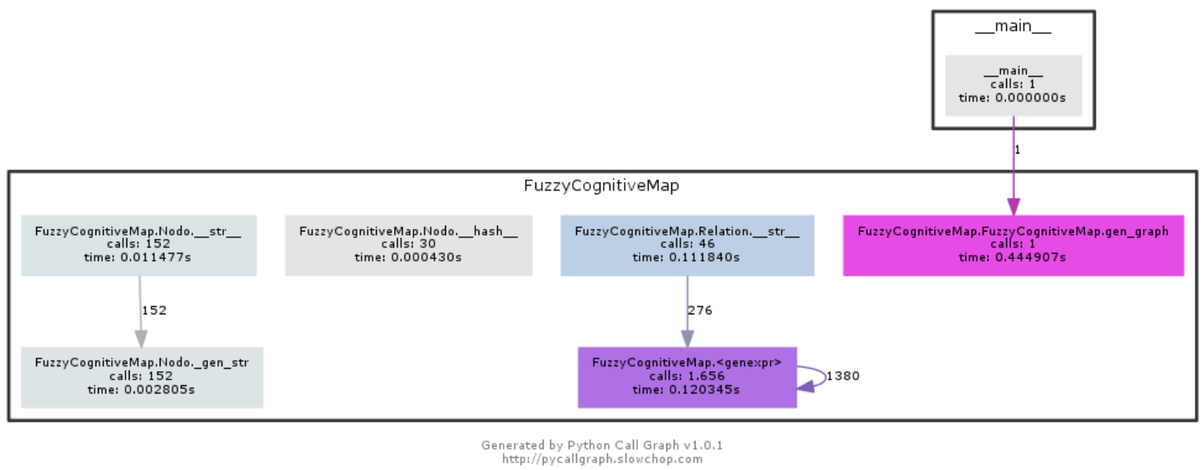

Figura I.1: Diagrama de llamadas del método *gen_graph*.

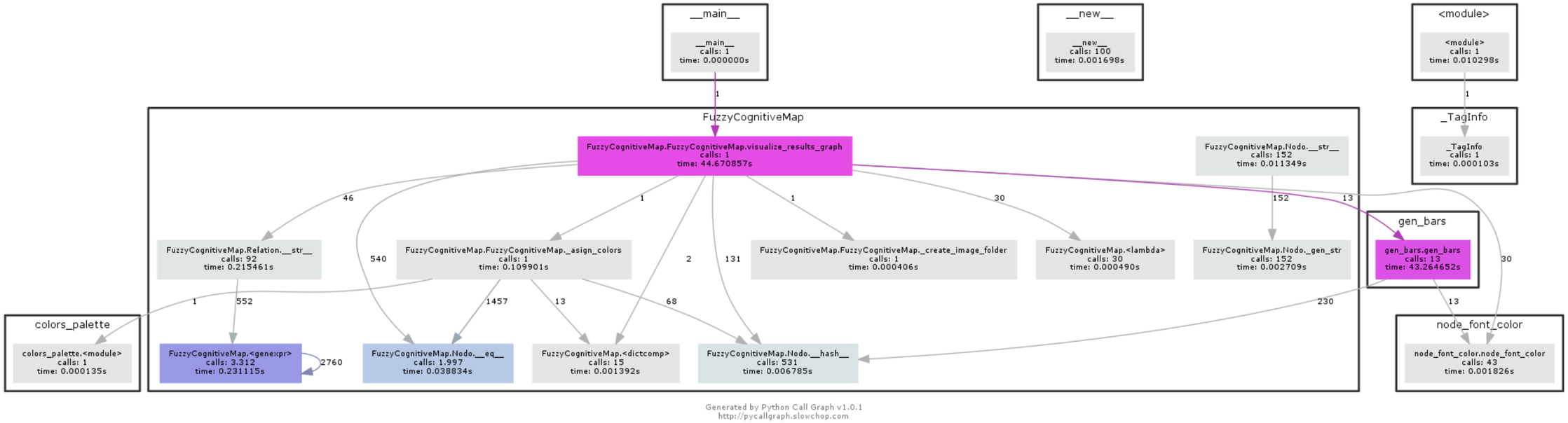

Figura I.2: Diagrama de llamadas del método *visualize_results_graph*.

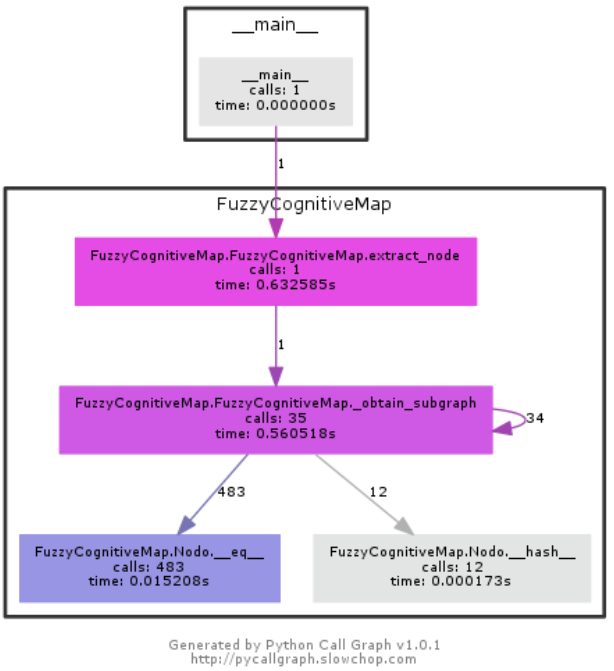

Figura I.3: Diagrama de llamadas del método *extract_node*.

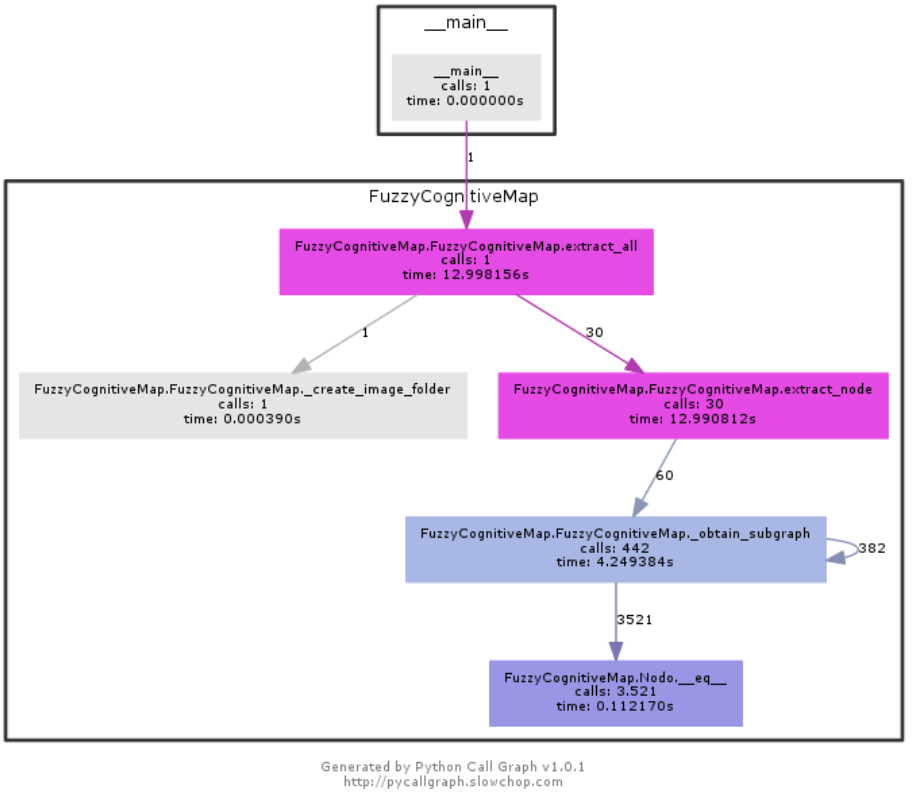

Figura I.4: Diagrama de llamadas del método *extract_all*.

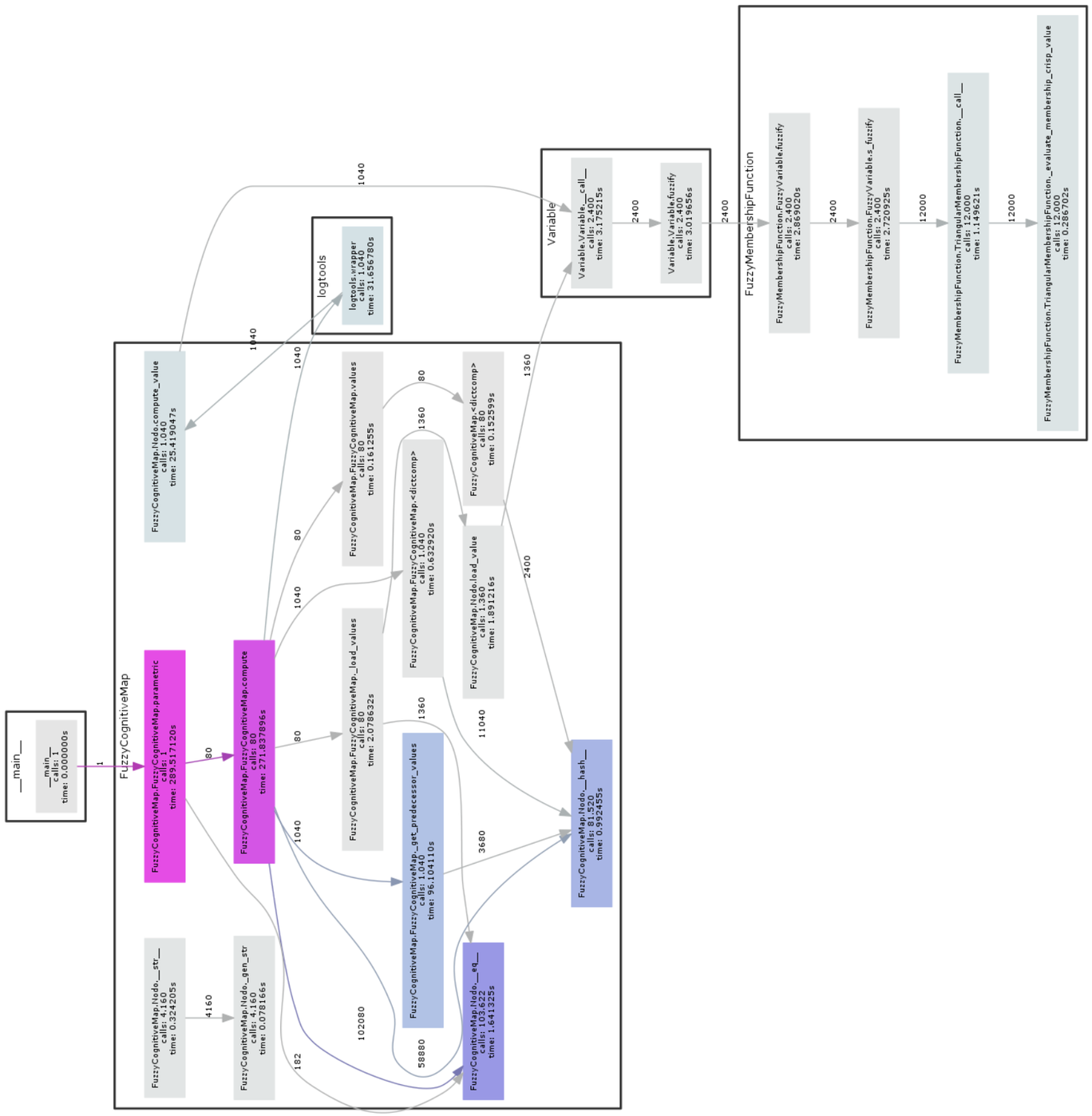

Figura I.5: Diagrama de llamadas del método *parametric*.

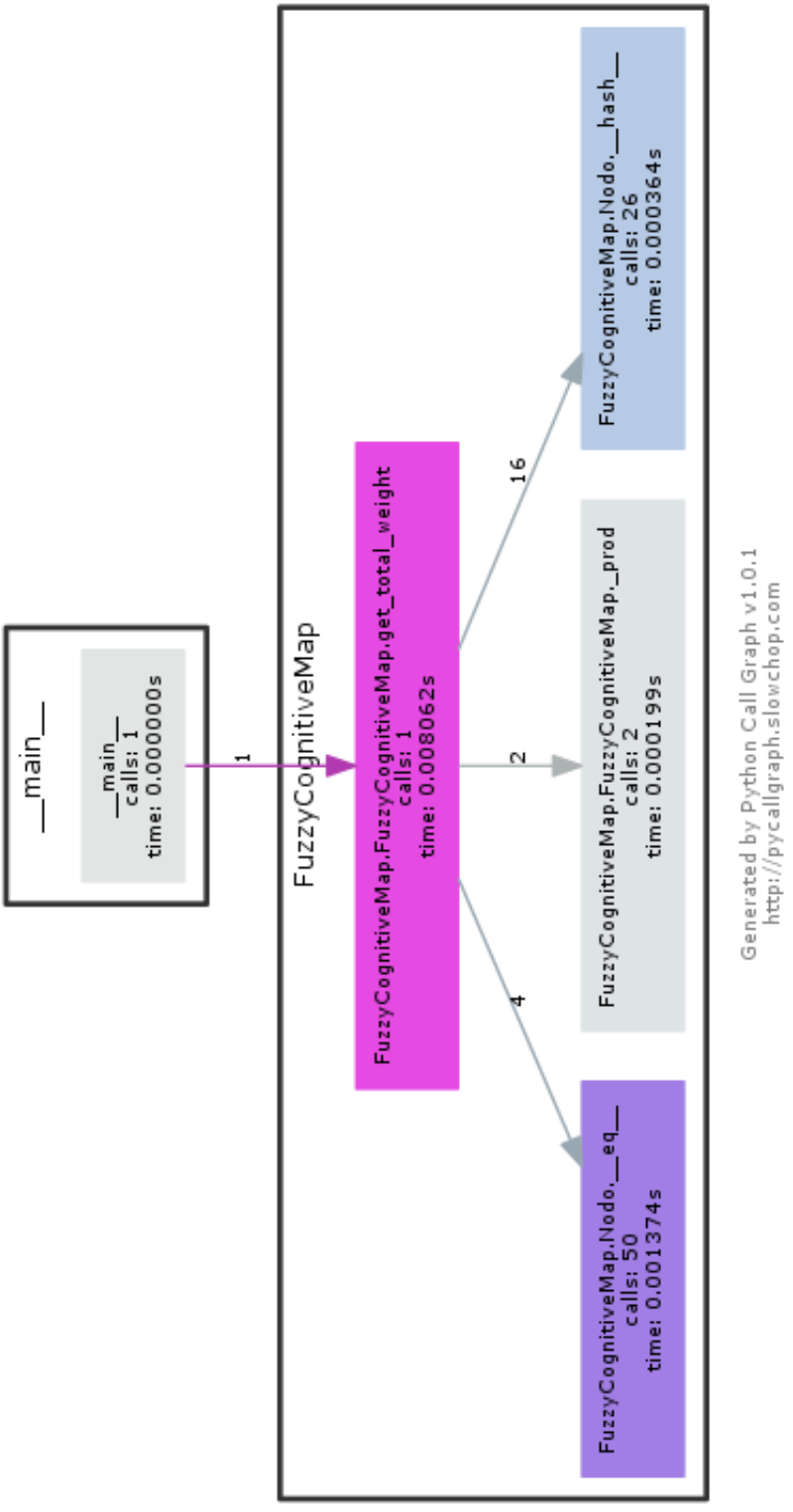

Figura I.6: Diagrama de llamadas del método *get_total_weight*.

Apéndice II

Modificaciones adicionales

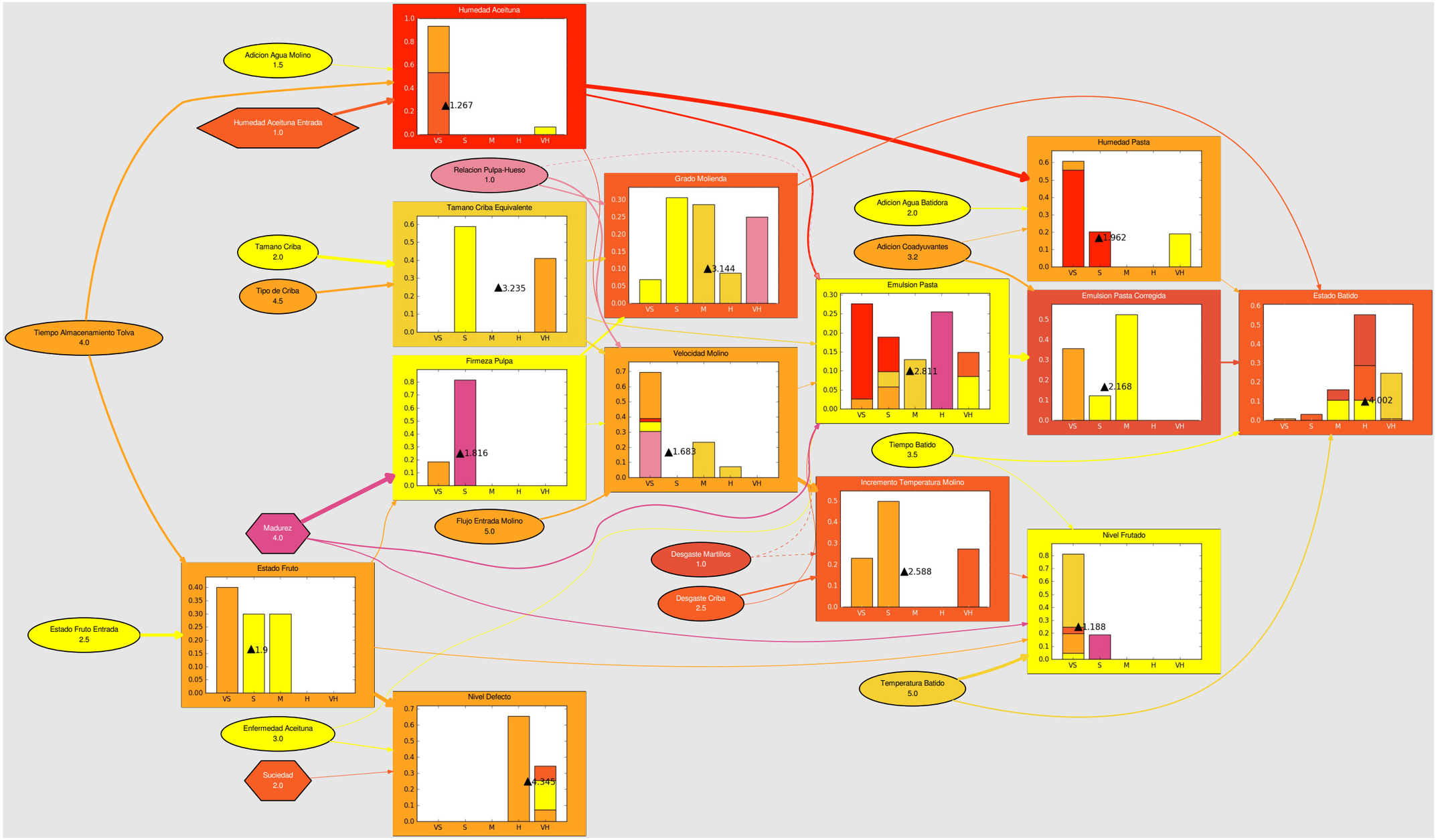

Figura II.1: Ejemplo 1: modelo de preparación de la pasta con paleta caliente.

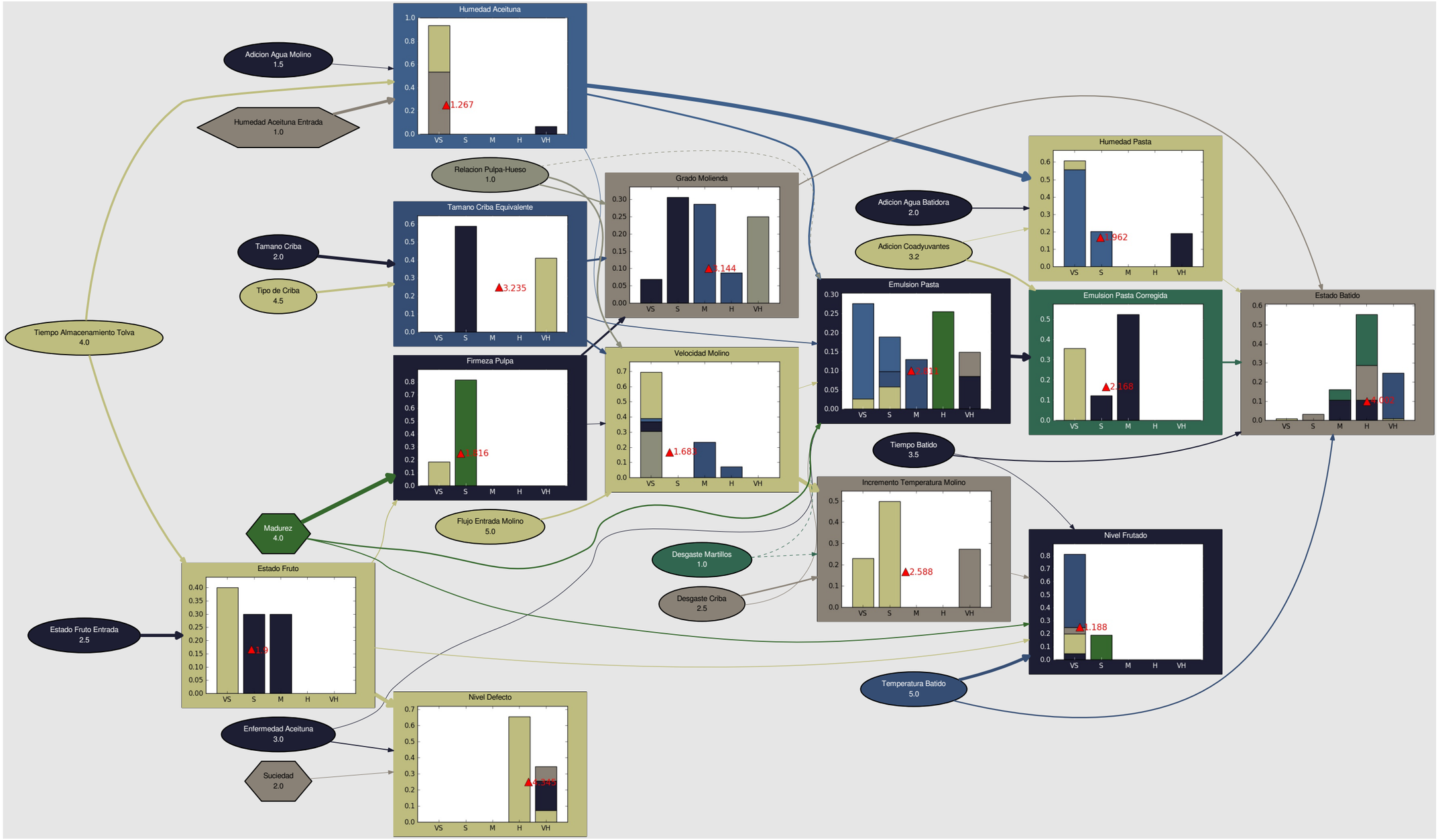

Figura II.2: Ejemplo 2: modelo de preparación de la pasta con paleta fria.

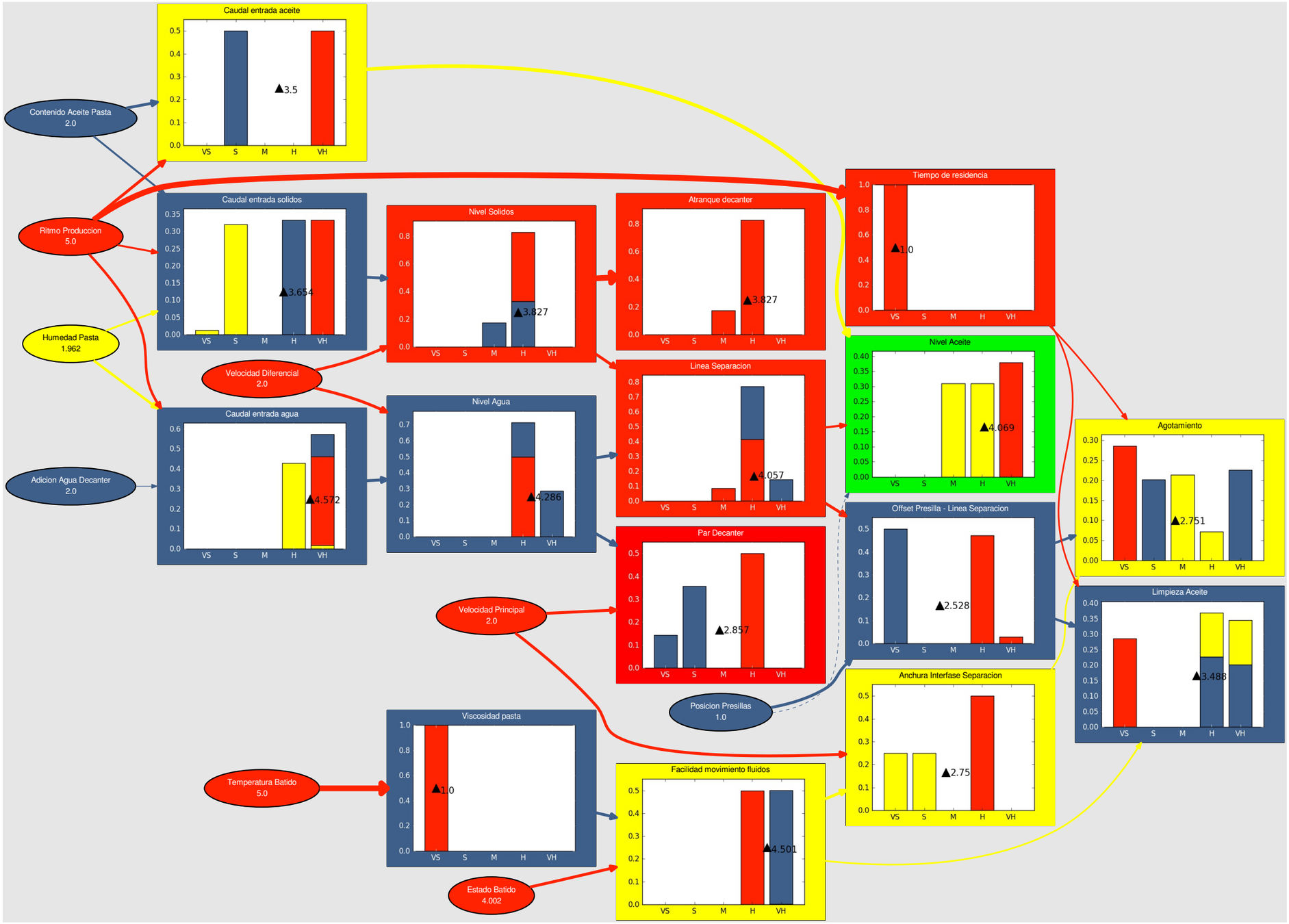

Figura II.3: Ejemplo 3: modelo de separación líquido-sólido con paleta brillante.

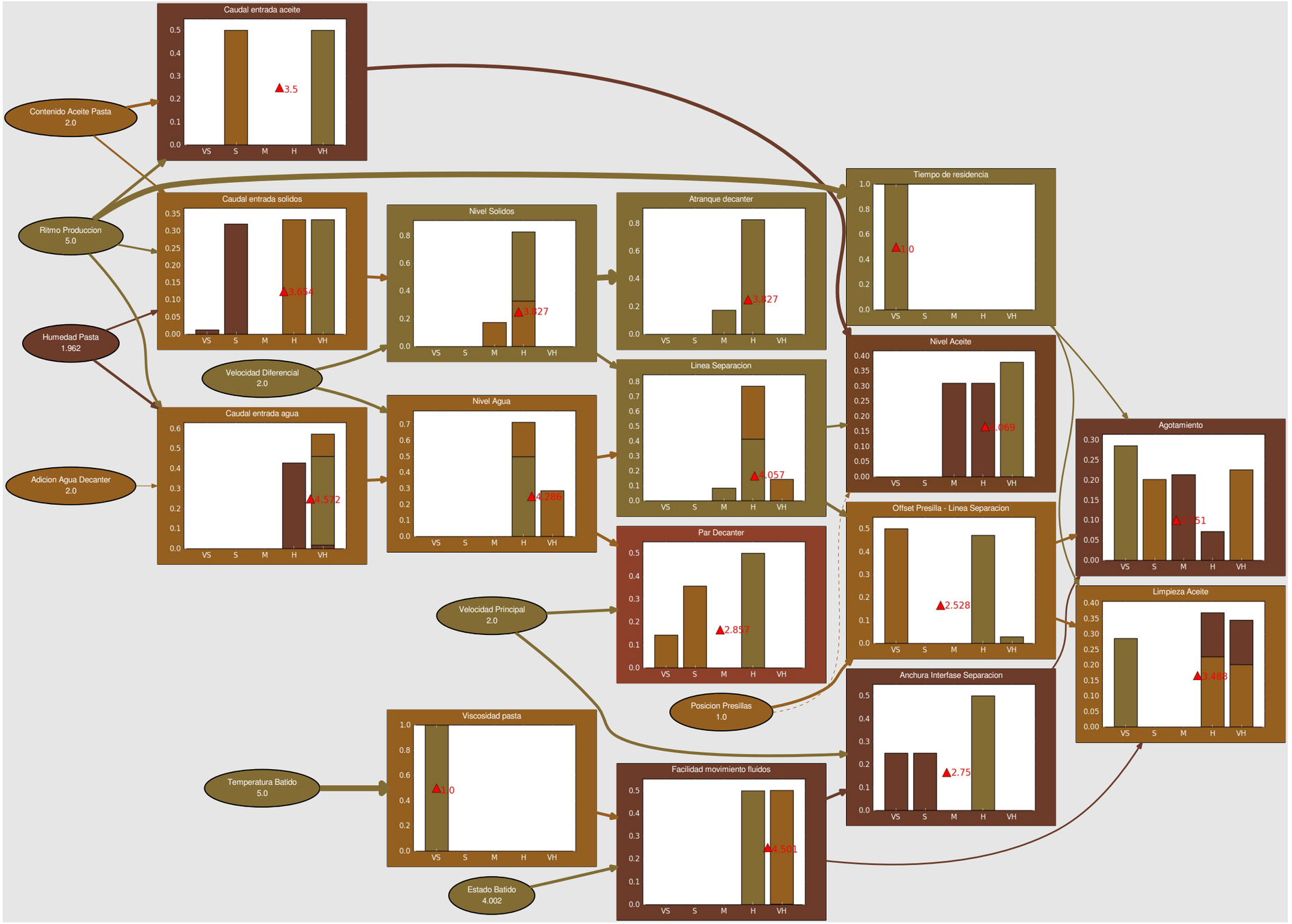

Figura II.4: Ejemplo 4: modelo de separación líquido-sólido con paleta otoñal.

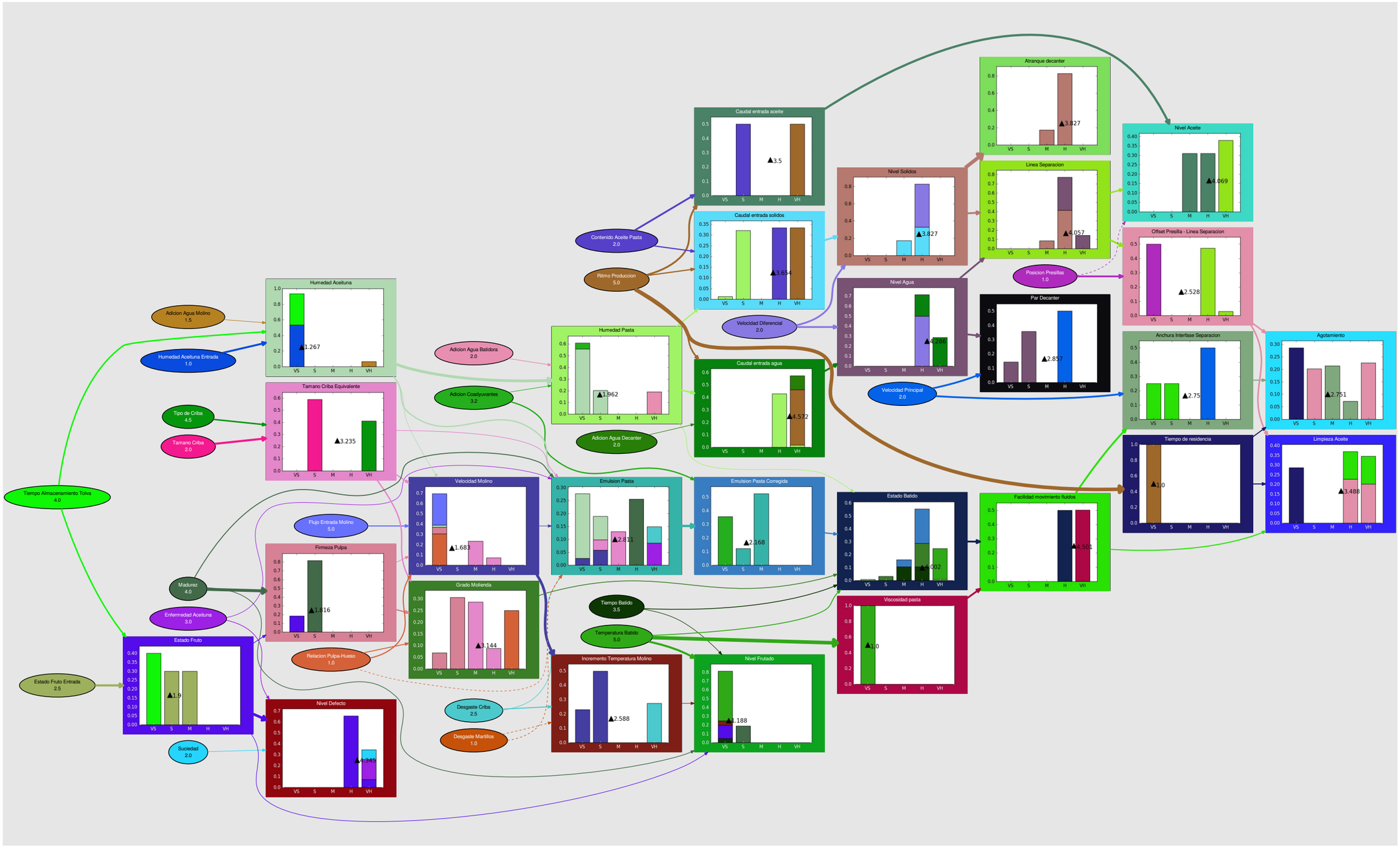

Figura II.5: Ejemplo 5: modelo conjunto con asignación aleatoria.

Apéndice III

Extracción de nodos

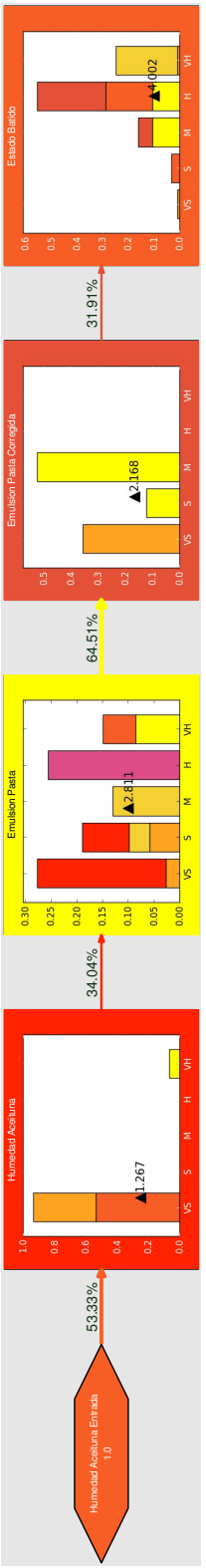

Figura III.1: Ejemplo 1: análisis *backward* de grado 1 de *Estado Batido*.

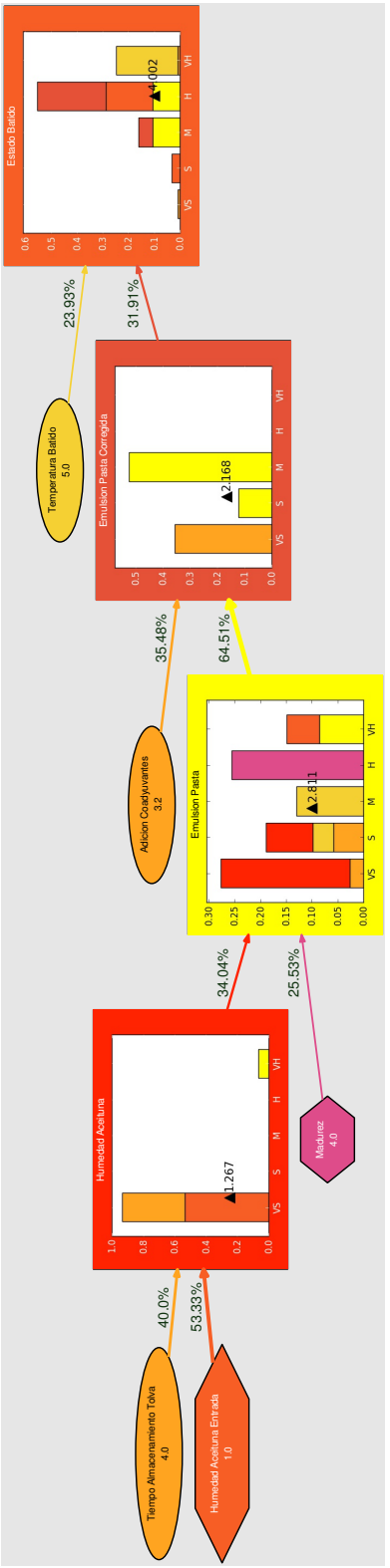

Figura III.2: Ejemplo 2: análisis *backward* de grado 2 de *Estado Batido*.

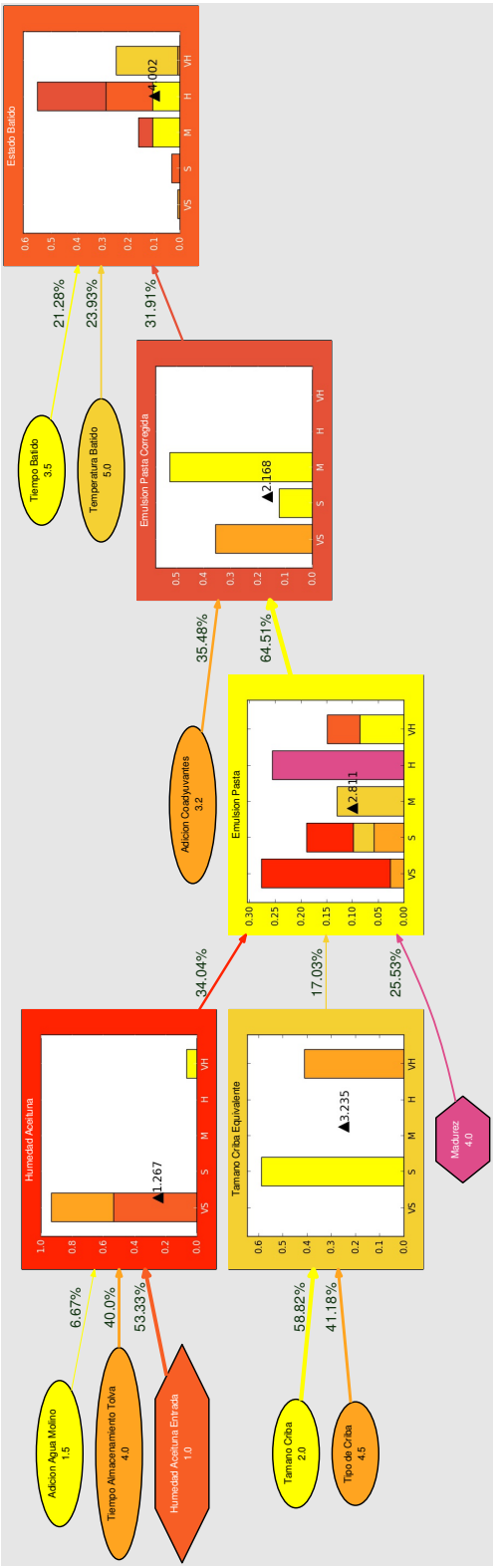

Figura III.3: Ejemplo 3: análisis *backward* de grado 3 de *Estado Batido*.

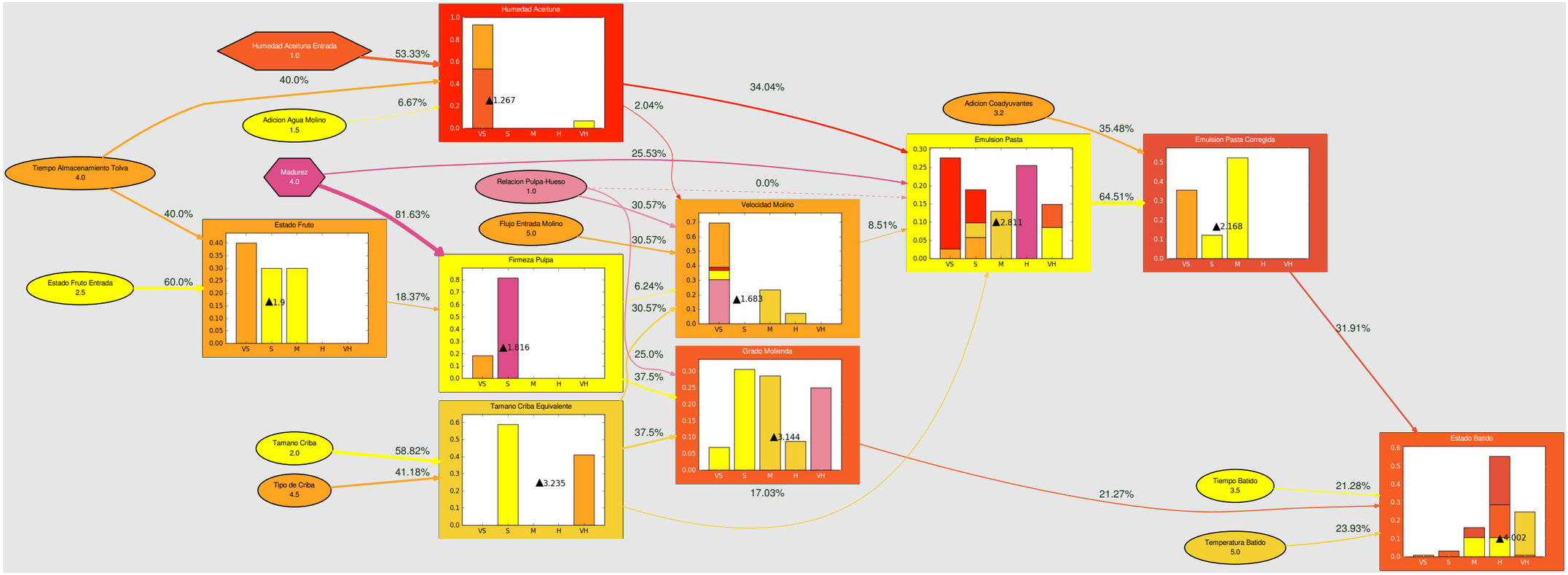

Figura III.4: Ejemplo 4: análisis *backward* de grado 4 de *Estado Batido*.

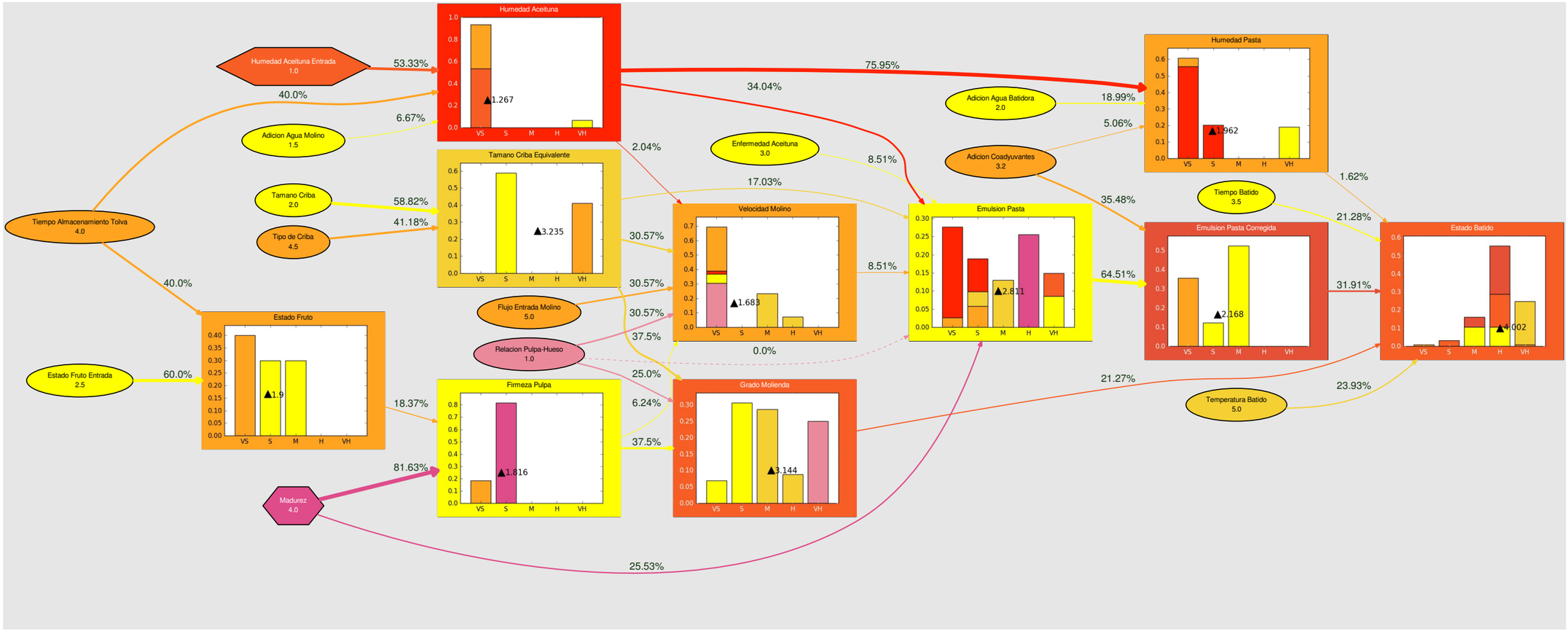

Figura III.5: Ejemplo 5: análisis *backward* de grado 5 de *Estado Batido*.

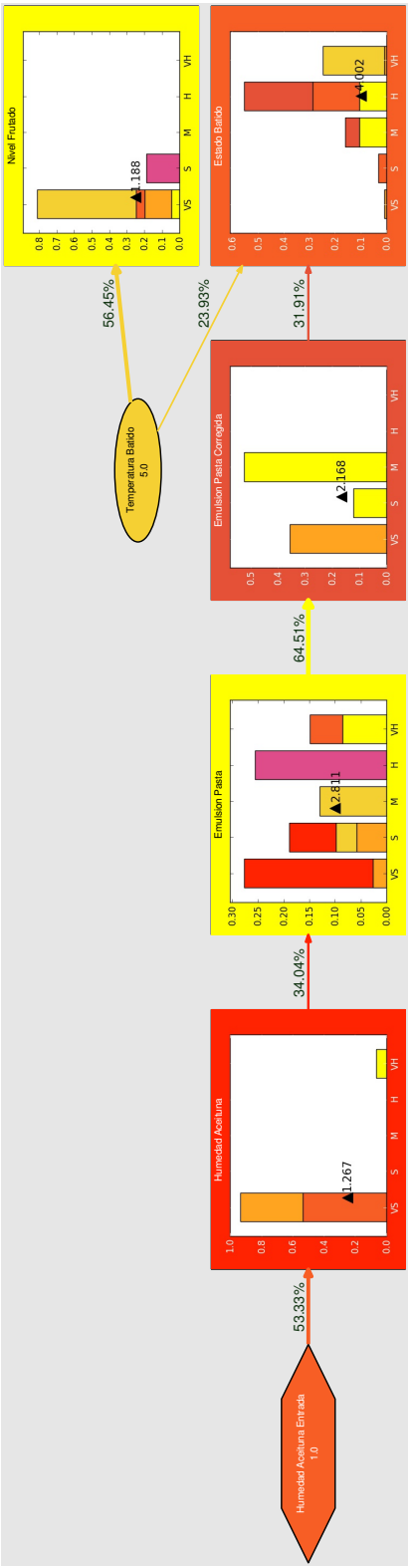

Figura III.6: Ejemplo 6: análisis *backward* de grado 1 de *Estado Batido* y *Nivel Frutado*.

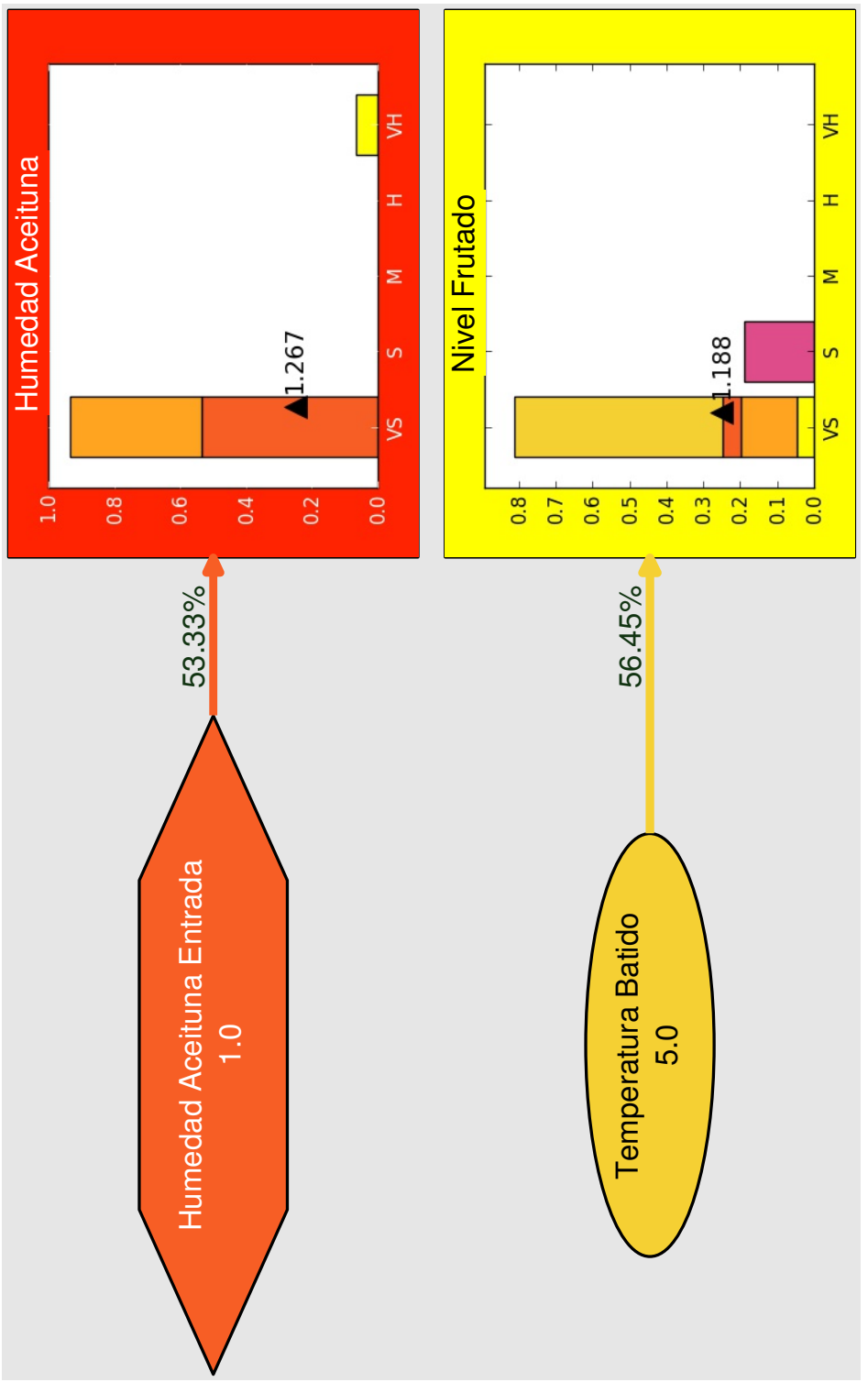

Figura III.7: Ejemplo 7: análisis *backward* de grado 1 de *Humedad Aceituna* y *Nivel Frutado*.

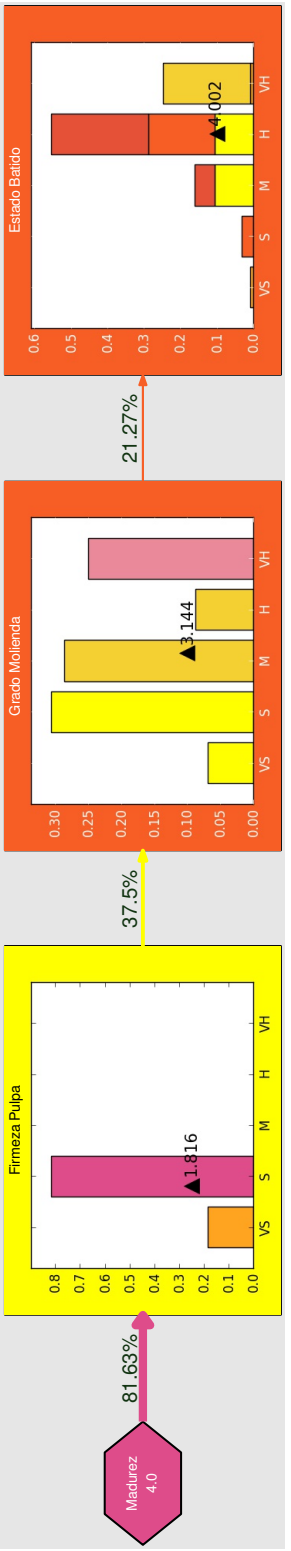

Figura III.8: Ejemplo 8: análisis *forward* de grado 1 de *Madurez*.

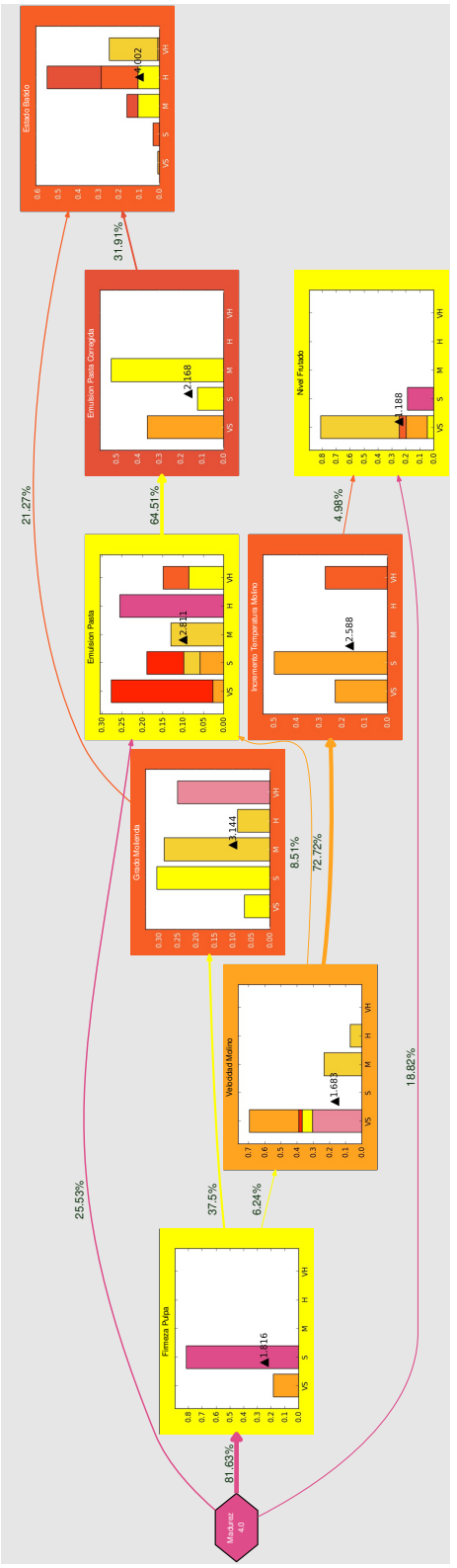

Figura III.9: Ejemplo 9: análisis *forward* de grado 2 de *Madurez*.

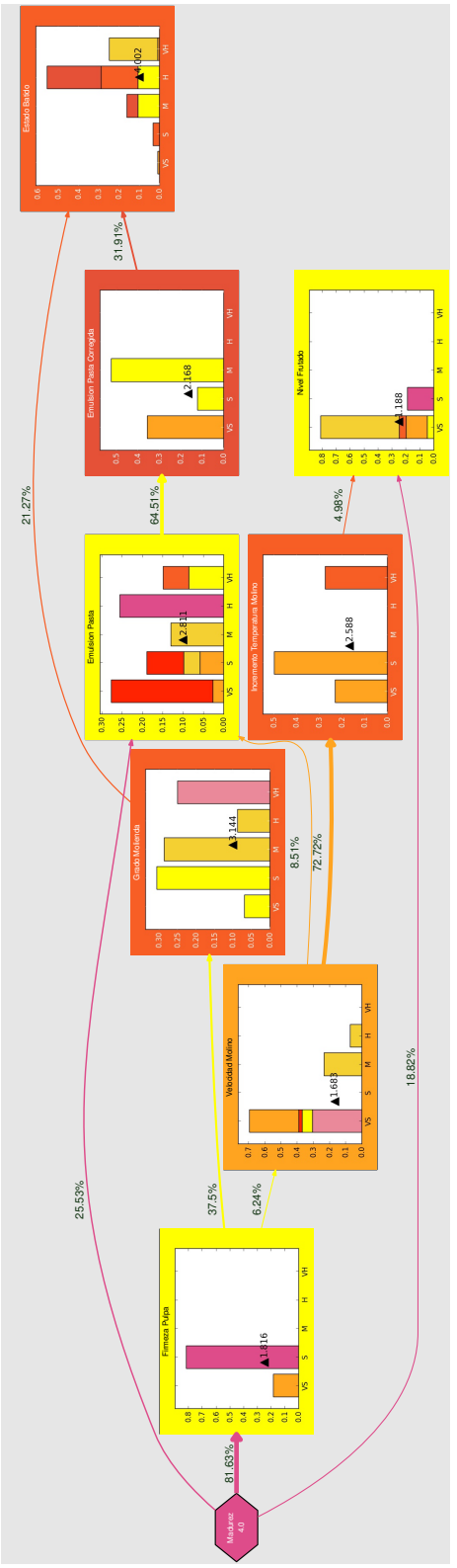

Figura III.10: Ejemplo 10: análisis *forward* de grado 3 de *Madurez*.

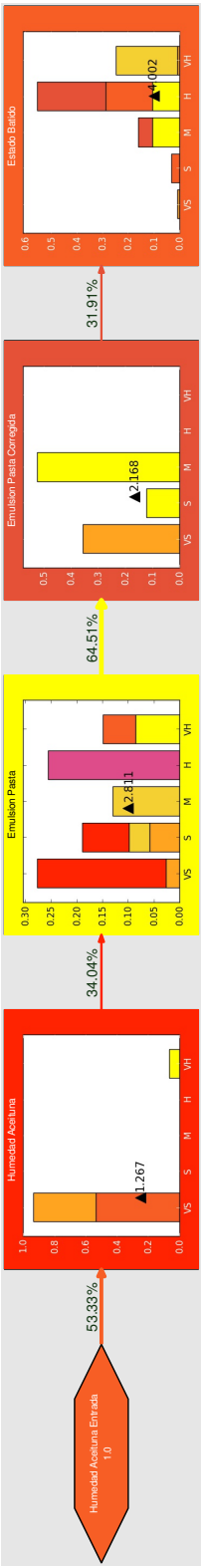

Figura III.11: Ejemplo 11: análisis *complete* de grado 1 de *Emulsion Pasta*.

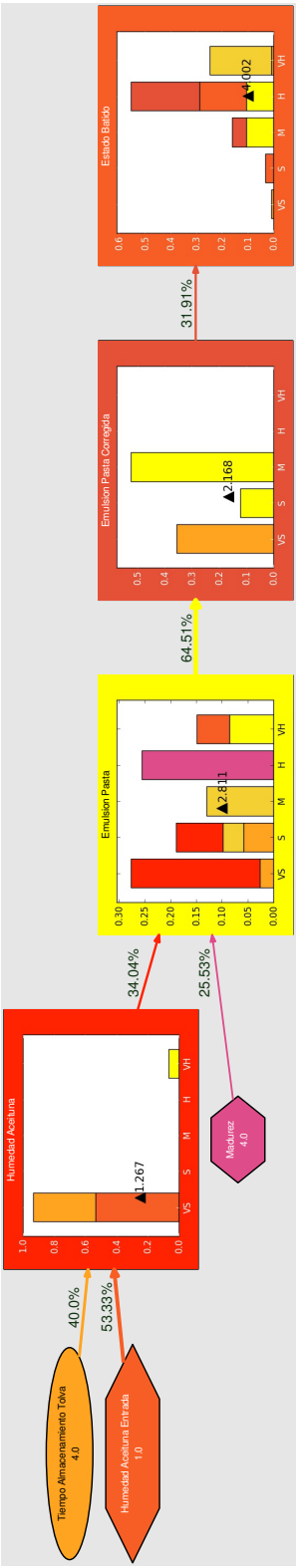

Figura III.12: Ejemplo 12: análisis *complete* de grado 2 de *Emulsion Pasta*.

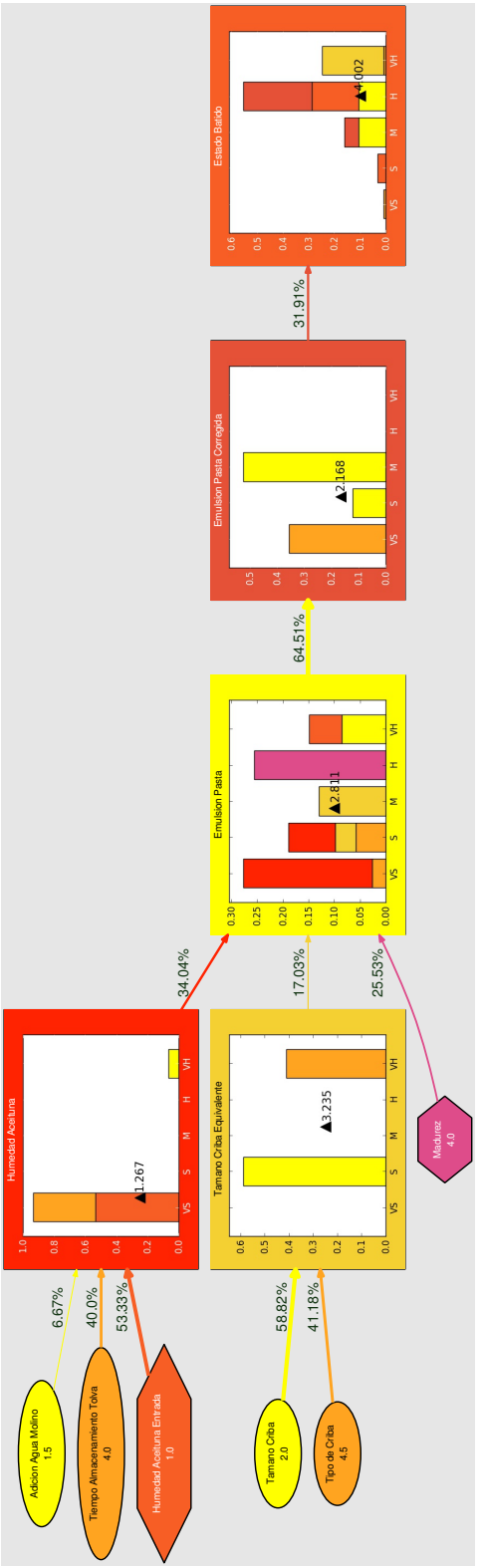

Figura III.13: Ejemplo 13: análisis *complete* de grado 3 de *Emulsion Pasta*.

Bibliografía

[Axe76] Robert Axelrod, editor. *Structure of Decision: The Cognitive Maps of Political Elites*. Princeton University Press, Princeton, N.J., 1st paperback edition edition edition, October 1976.

[BOBS89] Virginia E Barker, Dennis E O'Connor, Judith Bachant, and Elliot Soloway. Expert systems for configuration at digital: Xcon and beyond. *Communications of the ACM*, 32(3):298–318, 1989.

[DGdAM01] Raúl Pino Díez, Alberto Gómez Gómez, and Nicolás de Abajo Martínez. *Introducción a la inteligencia artificial: sistemas expertos, redes neuronales artificiales y computación evolutiva*. Universidad de Oviedo, 2001.

[FBCC+10] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

[FLB+13] David Ferrucci, Anthony Levas, Sugato Bagchi, David Gondek, and Erik T Mueller. Watson: beyond jeopardy! *Artificial Intelligence*, 199:93–105, 2013.

[HG08] Abdelwahab Hamam and Nicolas D Georganas. A comparison of mamdani and sugeno fuzzy inference systems for evaluating the quality of experience of hapto-audio-visual applications. In *Haptic Audio visual Environments and Games, 2008. HAVE 2008. IEEE International Workshop on*, pages 87–92. IEEE, 2008.

[Jac86] P. Jackson. *Introduction to expert systems*. Addison-Wesley Pub. Co.,Reading, MA, Jan 1986.

[Kos86] Bart Kosko. Fuzzy cognitive maps. *International Journal of Man-Machine Studies*, 24(1):65 – 75, 1986.

[LBFL80] Robert K Lindsay, Bruce G Buchanan, Edward A Feigenbaum, and Joshua Lederberg. Applications of artificial intelligence for organic chemistry: the dendral project. *New York*, 1980.

[Mal13] Jennifer L Malin. Envisioning watson as a rapid-learning system for oncology. *Journal of Oncology Practice*, 9(3):155–157, 2013.

[Mam74] Ebrahim H Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. In *Proceedings of the institution of electrical engineers*, volume 121, pages 1585–1588. IET, 1974.

[Mar15] Pablo Cano Marchal. *Contribution to the modeling and automatic control of the virgin olive oil elaboration process*. PhD thesis, Universidad de Jaén, 2015.

[MWGO16] P Cano Marchal, C Wagner, J Gámez García, and J Gómez Ortega. Modelling uncertainty in production processes using non-singleton fuzzification and fuzzy cognitive maps-a virgin olive oil case study. In *Fuzzy Systems (FUZZ-IEEE), 2016 IEEE International Conference on*, pages 1173–1180. IEEE, 2016.

[PS13] E.I. Papageorgiou and J.L. Salmeron. A review of fuzzy cognitive maps research during the last decade. *IEEE Transactions on Fuzzy Systems*, 21(1):66–79, February 2013.

[Sch83] Roger C Schank. *Dynamic memory: A theory of reminding and learning in computers and people*. cambridge university press, 1983.

[SDA+75] Edward H Shortliffe, Randall Davis, Stanton G Axline, Bruce G Buchanan, C Cordell Green, and Stanley N Cohen. Computer-based consultations in clinical therapeutics: explanation and rule acquisition capabilities of the mycin system. *Computers and biomedical research*, 8(4):303–320, 1975.

[TS85] Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE transactions on systems, man, and cybernetics*, (1):116–132, 1985.

[VV98] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, June 1965.

[Zad73] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(1):28–44, Jan 1973.