Estudio del problema del matrimonio estable en un entorno multiagente

Nombre Estudiante: Alberto Hueso Alonso
Grado Ingeniería Informática Especialidad Computación
Inteligencia Artificial

Nombre Consultor: *David Isern Alarcón*
Nombre Profesor responsable de la asignatura:*Carles Ventura Royo*

Fecha Entrega 02/01/2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Estudio del problema del matrimonio estable en un entorno multiagente*
Nombre del autor:	*Alberto Hueso Alonso*
Nombre del consultor/a:	*David Isern Alarcón*
Nombre del PRA:	*Carles Ventura Royo*
Fecha de entrega (mm/aaaa):	*01/2018*
Titulación o programa:	*Grado de Ingeniería Informática* *Especialidad Computación*
Área del trabajo final	*Inteligencia artificial*
Idioma del trabajo:	*Castellano*
Palabras clave:	*Sistemas multiagente, matrimonio estable, clasificadores*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

El presente trabajo pretende determinar si un algoritmo de emparejamiento estable, como el de Gale-Shapley, puede ser sustituido por un sistema supervisado de clasificación de forma que encuentre también parejas estables de una forma más eficiente.

Para ello, se ha desarrollado un software basado en un sistema multi agente implementado con JADE. El sistema genera, en forma de agentes, el mismo número de elementos de dos grupos diferentes. Cada agente nace con una serie de características, se comunica con todos los agentes del grupo contrario y los ordena, según sus propias preferencias en una lista de parejas favoritas. Envía esta lista a otro agente, un emparejador, que utiliza el algoritmo de Gale-Shapley para formar parejas estables y las guarda en archivos CSV. Un tercer tipo de agente, un clasificador supervisado de las bibliotecas Weka (Logistic), es entrenado con esos datos y es evaluado para comprobar su grado de acierto.

Se ha encontrado que el clasificador es capaz de encontrar algunas parejas estables pero lo hace en un grado muy escaso y con una precisión también demasiado pequeña.

No se justifica, por tanto el uso de un clasificador para sustituir el algoritmo de Gale-Shapley.

Abstract (in English, 250 words or less):

The present work tries to find out if a stable pairing algorithm, such as the Gale-Shapley , can be replaced by a supervised classification system so that it also finds stable pairs in a more efficient way.

To achieve this, we have developed a software based on a multi agent system implemented with JADE. The system generates, as agents, the same number of elements from two different groups. Each agent is created with a series of characteristics, communicates with all the agents from the opposite group and orders them, according to their own preferences, in a list of favorite couples. Later, it sends this list to another agent, a matchmaker, which uses the Gale-Shapley algorithm to form stable pairs and saves them in CSV files. A third type of agent, a supervised classifier from the Weka libraries, Logistic, is trained with this data and is evaluated to check its degree of success.

We have seen this classifier is able to find some stable pairs but just a few ones, and with a very small precision too.

It is not justified, therefore, the use of a classifier to replace the Gale-Shapley algorithm.

Índice

Lista de figuras

1 Introducción

1.1 Contexto y justificación del Trabajo

1.1.1 Temática

Desde que comencé a estudiar informática, siempre me han interesado especialmente tres ideas:

- Los algoritmos de resolución de problemas genéricos pero aplicables a diversos problemas concretos del mundo real, solo en apariencia muy diferentes.
- Los sistemas multiagente, que combinan la acción de diferentes agentes autónomos, una poderosa herramienta aplicable en toda clase de ámbitos.
- Las diversas técnicas de aprendizaje computacional, en especial los clasificadores supervisados, que permiten a un sistema acumular conocimiento y utilizarlo posteriormente para realizar predicciones.

A la hora de escoger el trabajo de fin de grado, he tratado de conjugar esos tres conceptos, estudiando un algoritmo de resolución de un problema, en un entorno multiagente que simulara el funcionamiento del algoritmo y suministrando, a partir de los datos obtenidos en la simulación, información a clasificadores que, una vez entrenados, trataran de predecir el resultado del algoritmo de una forma, posiblemente, más eficiente.

Como algoritmo de estudio me he fijado en el problema del **matrimonio estable** [1], que forma parte de los problemas de emparejamiento. Consiste en emparejar todos los elementos de dos grupos, teniendo en cuenta que cada elemento tiene preferencias respecto a con qué miembros del otro grupo quiere estar asociado. El objetivo es asegurar que las parejas formadas sean estables, esto es, todas las parejas están en una situación en la que ningún miembro de las parejas constituidas prefiera a alguien de otra pareja y éste a su vez le prefiera a él también antes que el elemento con quien está emparejado.

Este problema, en apariencia simple, está en la base de multitud de situaciones como asignar médicos residentes a hospitales, repartir estudiantes de un colegio mayor en diferentes habitaciones, decidir qué *cluster* del CDN se utilizará para descargar contenido [2] , etc...

Se han desarrollado diferentes algoritmos para dar resolución a este tipo de problemas, siendo quizá el más famoso el creado, ya en 1962, por David Gale y Lloyd Shapley para asignar estudiantes a universidades[3].

Desde entonces, otros algoritmos tratan de superar su eficiencia [4] o incluso alcanzar soluciones que además de la estabilidad buscan optimizar la satisfacción de las parejas formadas [5].

En cuanto a los sistemas multiagente (SMA)[6] , se ha decidido adoptar JADE [7], *JavaAgent Develpment Environment*, un *framework* que simplifica la implementación de agentes según las especificaciones de la *Foundation for Intelligent Physical Agents* (FIPA)[8] .

Como clasificadores, se ha optado por utilizar el conjunto de bibliotecas Weka[9] que permiten integrarse con JADE.

Por último, como entorno de programación se ha elegido Eclipse [10].

1.1.2 Problemática a resolver

Tal y como se ha avanzado, en la literatura científica hay innumerables algoritmos que dan solución al problema del matrimonio estable, en ocasiones tratan de mejorar su eficiencia [4], en otras mejorar otros aspectos como pudiera ser la justicia de los emparejamientos [5]. De todos ellos este trabajo se centrará en el algoritmo original, el de Gale-Shapley, que requiere que se examinen todos y cada uno de los miembros de los dos conjuntos a emparejar de manera que si el grupo es muy grande el tiempo de resolución podría ser excesivo.

La pregunta que trata de responder este trabajo es si un sistema de aprendizaje supervisado, una vez entrenado, podría llegar a clasificar un potencial emparejamiento sin necesidad de ejecutar el algoritmo completo, sin examinar el resto de parejas de una manera más eficiente y con un grado de fiabilidad aceptable.

Para ello se propone, en primer lugar una simulación mediante agentes.
El sistema crea un número, elegido por el usuario de agentes de dos tipos, X e Y. Cada agente tendrá una serie de atributos y una serie de preferencias respecto a los atributos del otro grupo.

Cada agente se comunica con todos los agentes del grupo contrario intercambiándose sus atributos y los valora, en este caso multiplicando el valor de una característica por su propia preferencia para esa misma característica y las sumará. Posteriormente ordena de mayor a menor todos los agentes del grupo contrario en una lista de agentes preferidos.

Cuando un agente completa su lista de preferencias, ya ordenada, la envía a otro agente del sistema, un emparejador, quien cuando tiene todas las listas ejecuta el algoritmo de matrimonio estable, formando parejas y deshaciéndolas cuando sean no estables hasta que todas sean estables.

Esto forma parte únicamente de la simulación, pero un nuevo tipo de agentes, los clasificadores pueden ser entrenados con la información de las parejas formadas categorizándolas según sean estables o no estables.

De esta forma el clasificador, o clasificadores, una vez recibidos suficientes datos podrían intentar predecir si una nueva pareja será estable o no. Se podrá evaluar la tasa de aciertos comparando con el resultado del algoritmo y, comparando también los tiempos de procesado, se podrá determinar si es factible sustituir el algoritmo completo por las predicciones del clasificador.

1.2 Objetivos del trabajo

Los objetivos del TFG se han establecido en forma de hitos a lograr a lo largo de la elaboración del mismo.

- Conocer en profundidad el *framework* JADE
- Crear agentes de dos grupos, les llamaremos X e Y, con una serie de atributos y preferencias, en principio, aleatorias.
- Ser capaz de comunicar los agentes de un grupo con los del otro, transmitiendo la información de sus atributos.
- Conseguir que todos los agentes de cada grupo contacten con los agentes del otro grupo.
- Desarrollar algoritmo para ordenar según las preferencias de un agente las potenciales parejas del grupo contrario.
- Implementar la comunicación entre los agentes con el emparejador, transmitiendo las listas de parejas ordenadas por preferencia.
- Implementar el algoritmo de matrimonio estable en el emparejador. Alternativamente, vincular a través de Java con el paquete de software estadístico R que ya lo tiene implementado.
- Vincular el SMA con las bibliotecas Weka.
- Crear, como mínimo, un agente clasificador. El método de clasificación está en la fase de planificación por determinar.
- Comunicar el emparejador con los agentes clasificadores. La información transmitida serán parejas formadas, con sus atributos, y la categoría a la que pertenecen, estables o no.
- Entrenar a los clasificadores con un conjunto de entrenamiento de cardinalidad inicialmente por determinar.
- Comparar los resultados del algoritmo de matrimonio estable con las predicciones de los clasificadores en un conjunto de prueba.
- Elaborar conclusiones respecto a la eficiencia y eficacia de la utilización de los clasificadores en comparación al algoritmo de Gale-Shapley.

1.3 Enfoque y modo seguido

Como ya se ha señalado, hay diferentes algoritmos que implementan el problema del matrimonio estable. Una de las primeras decisiones a tomar es elegir cuál de ellos se utilizará. Ya se ha avanzado también que se ha elegido el de Gale-Shapley, tanto por su relevancia histórica como por su simplicidad.

Existen dos posibles estrategias para comparar el algoritmo de Gale-Shapley con clasificadores.

La primera es buscar y encontrar estudios con tablas de datos de parejas ya estables y alimentar con ellos uno o varios clasificadores.

La segunda, es crear esas parejas estables con una simulación mediante agentes que se emparejen entre sí.

Aunque la primera opción es más sencilla, rápida y partiendo de datos reales proporcionaría un resultado de más utilidad inmediata, se ha optado por el enfoque de la simulación previa porque nos permite profundizar en diferentes aspectos, por ejemplo, la comunicación entre agentes, que son también objetivos personales muy importantes de este trabajo.

1.4 Planificación del Trabajo

1.4.1 Planificación inicial

WBS	Nombre	Inicio	Fin	Trabajo	Terminado
1	Comienzo	Sep 20	Sep 20		
2	PAC0	Sep 20	Oct 2	10d	
2.1	Estudio de ejemplos previos	Sep 20	Sep 22	3d	100%
2.2	Selección de propuestas	Sep 25	Sep 28	4d	100%
2.3	Acuerdos con consultor	Sep 28	Oct 2	2d	100%
2.4	Elaboración y entrega PAC0	Sep 29	Oct 2	1d	100%
2.5	FIN PAC0	Oct 2	Oct 2		
3	PAC1	Oct 3	Oct 16	10d	
3.1	Planificación en papel	Oct 3	Oct 5	3d	100%
3.2	Instalación y	Oct 6	Oct 6	1d	100%

	repaso herramienta Gantt				
3.3	Elaboración diagrama Gantt	Oct 9	Oct 13	5d	100%
3.4	Integrar en memoria	Oct 16	Oct 16	1d	100%
3.5	FIN PAC1	Oct 16	Oct 16		
4	PAC2	Oct 17	Nov 20	29d	
4.1	Estudio a fondo framework JADE	Oct 17	Oct 23	5d	100%
4.2	Creación y comunicación agentes	Oct 24	Oct 30	5d	
4.2.1	Creación agentes	Oct 24	Oct 25	2d	100%
4.2.2	Comunicación agentes	Oct 26	Oct 30	3d	100%
4.3	Completar relaciones entre agentes	Oct 31	Nov 6	5d	100%
4.4	Implementar algoritmo ordenar preferencias	Nov 7	Nov 10	4d	100%
4.5	Comunicar preferencias Agent Manager	Nov 13	Nov 14	2d	100%
4.6	Elaboración PAC2	Nov 15	Nov 20	4d	100%
4.7	Reajuste Planificación	Nov 15	Nov 20	4d	100%
4.8	FIN PAC2	Nov 20	Nov 20		
5	PAC3	Nov 21	Dec 18	30d	
5.1	Implementar matrimonio estable en Agent Manager	Nov 21	Nov 27	5d	100%
5.2	Vincular SMA con clasificadores Weka	Nov 21	Nov 27	5d	100%
5.3	Implementar entrenamiento clasificador	Nov 28	Dec 4	5d	100%
5.4	Estudio de	Dec 5	Dec 11	5d	100%

	resultados				
5.5	Elaboración PAC3	Dec 12	Dec 18	5d	100%
5.6	Reajuste Planificación	Dec 12	Dec 18	5d	100%
5.7	FIN PAC3	Dec 18	Dec 18		
6	PAC4	Dec 18	Jan 2	11d	
6.1	Memoria	Dec 19	Jan 2	11d	100%
6.2	Fin PAC4	Jan 2	Jan 2		
7	PAC5	Jan 3	Jan 22	14d	
7.1	PAC5a Presentación	Jan 3	Jan 10	6d	0%
7.2	PAC5b Defensa	Jan 11	Jan 22	8d	0%
8	ELABORACIÓN MEMORIA	Sep 20	Jan 2	75d	0%
9	FIN DE PROYECTO	Jan 22	Jan 22		

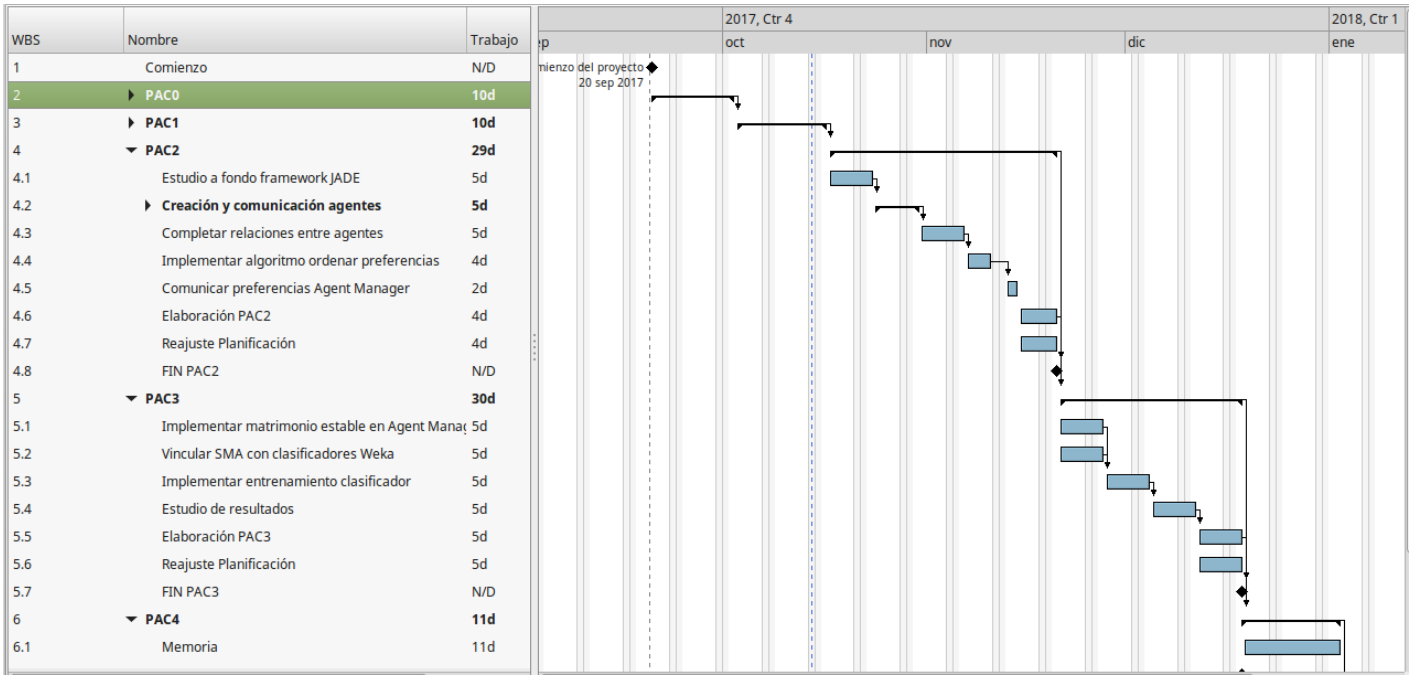

Fig. 1: Diagrama de Gantt

Riesgo	Contramedida
Pérdida de datos	Copias de seguridad en Dropbox.
Avería equipo	Utilización de otro equipo, se dispone de dos.
Incumplimiento de plazos	Dedicación de tiempo adicional. La planificación inicial no prevé trabajo en fin de semana, se pueden utilizar esos días
Imposibilidad de utilizar alguno de los entornos previstos para la tarea prevista	Sustitución de JADE por otro SMA Sustitución de Weka por otra biblioteca estadística o, en el peor de los casos, adaptación del clasificador implementado en la asignatura de Aprendizaje Computacional.

1.5 Breve sumario de productos obtenidos

- Este documento, la memoria del trabajo.
- Proyecto Eclipse con:
  - Paquete tfgMarriageStable con el software desarrollado para realizar el estudio.
  - Paquete pruebasGaleShapley con un sofware independiente que implementa el algoritmo de Gale-Shapley y su comprobación.
  - Documentación de las diferentes clases en formato HTML.
- Archivos elementosX.out elementosY.out y parejas.txt, que se utilizarán para comprobar el funcionamiento de este paquete de forma independiente.
- Archivos resultados.ods y tiemposEmparejamientos.ods con los datos recopilados para realizar el estudio.
- Archivos de ejemplos del resultado de unos emparejamientos.
  GaleShapleyCompleto_parejas_200_Iteraciones_1_2017_12_10_15_49_38modulacion:true.csv
  GaleShapley_parejas_200_Iteraciones_5_2017_12_10_17_36_16modulacion:true.csv

1.6 Breve descripción de los otros capítulos de la memoria

2. Desarrollo del trabajo: Esta sección constituye el núcleo del trabajo. En ella se describe qué se ha realizado, qué metodologías se han empleado y qué resultados se han obtenido.

2.1 Planteamiento del problema:Describe la historia del problema del matrimonio estable, un ejemplo de su significado y explica el algoritmo que desarrollaron Gale y Shapley para darle solución.

2.2 Entornos y software utilizados: Describe qué elementos se han utilizado para el desarrollo del sofware que se ha creado para realizar el estudio. Estos han sido Eclipse, JADE y Weka.

2.3 Implementación: Explica en detalle cuál ha sido la implementación de dicho software. Cómo funciona, qué clases se han creado y para qué se utilizan, así como las decisiones más importantes del diseño.

2.4 Estudio de resultados: Analiza los resultados obtenidos tras varias ejecuciones del programa creado para determinar si el algoritmo de Gale-Shapley puede ser sustituido por un clasificador.

3 Conclusiones: Analiza de una forma más general los resultados obtenidos, tratando de explicar las causas de los mismos. Asimismo estudia si se han conseguido los objetivos planteados inicialmente, la metodología empleada y, en definitiva, la calidad del trabajo. Por último, señala algunas posibles líneas de ampliación futuras.

4 Glosario: Define algunos términos con los que el lector puede no estar familiarizado.

5 Bibliografía: Señala las fuentes de información en las que se ha basado el trabajo.

6 Manual de uso: Describe los requerimientos, instalación y como utilizar el software desarrollado, tanto en Linux como en Windows, sus opciones de uso y cómo importarlo y ejecutarlo también en Eclipse.

2 Desarrollo del trabajo

2.1 Planteamiento del problema

2.1.1 Historia [1]

Existen numerosas situaciones en las que se han de emparejar diferentes elementos pertenecientes a grupos distintos. Cuando estos elementos pueden tomar decisiones, establecen preferencias respecto a con qué elementos de los otros grupos quieren estar asociados y, en consecuencia, prefieren a unos respecto a otros.

Esto plantea el problema de cómo realizar estos emparejamientos de forma que permanezcan estables, esto es, sin que dos elementos rompan sus respectivas parejas para formar una nueva porque se prefieren entre sí antes que a aquellos con los que estaban anteriormente.

Históricamente, esta dificultad se hizo patente en Estados Unidos cuando, mediado el siglo XIX, la gran competencia entre estudiantes residentes por elegir hospital y la falta de regulación propició que los estudiantes fueran cambiando de centro a medida que recibían ofertas de hospitales que les satisfacían más. Los hospitales por su parte elegían estudiantes al poco de empezar su formación, sin saber si se adecuarían a sus intereses. Todo ello generaba serios problemas.

Sería necesario esperar hasta 1951, cuando se puso en marcha el NRMP (Programa Nacional de Emparejamiento de Residentes) que se aplicó un algoritmo satisfactorio para unir a los estudiantes y los hospitales según sus respectivas preferencias.

Algunos años después, en 1962, D. Gale y L. Shapley, desconociendo el algoritmo del NRMP, resolvieron el problema que subyace en la asignación de residentes/hospitales: el problema del matrimonio estable, permitiendo su generalización y su uso en otros entornos. Hoy en día se usan en ámbitos tan diferentes como la asignación de riñones a enfermos para ser trasplantados [11], o decidir qué *cluster* del CDN se utilizará para descargar contenido [2].

2.1.2 Problema del matrimonio estable, un ejemplo [1]

Sin entrar en formalismos, la mejor forma de entender el problema es probablemente mediante un ejemplo.

Supongamos que se tiene un grupo de hombres y otro de mujeres, cada uno de cuyos miembros desea establecer una relación con un miembro del otro grupo.

Sean los hombres: {Kepa, Jordi, Antonio}; y las mujeres:{Eva, Montse, Judith}

Cada uno de ellos tiene las siguientes listas de preferencias, ordenadas de mayor a menor. Por ejemplo, la favorita de Kepa es Judith, seguida de Montse y por último Eva.

Kepa→(Judith, Montse, Eva)	Eva →(Antonio,Kepa, Jordi)
Jordi→(Montse, Judith, Eva)	Montse→(Jordi,Antonio,Kepa)
Antonio→(Montse,Eva, Judith)	Judith→(Kepa,Antonio,Jordi)

Supongamos que se forman las siguientes parejas:

a=Jordi y Montse, **b**=Kepa y Eva, **c**=Antonio y Judith.

La pareja **a** es claramente estable ya que sus dos miembros son el favorito del otro, por lo que ninguno tiene ningún interés en cambiar.

Las parejas **b** y **c**, en cambio, son inestables porque Kepa prefiere a Judith antes que a Eva y Judith, a su vez, también prefiere a Kepa sobre Antonio con quien está.

Sin embargo, una vez que se produce ese cambio, todas las parejas serán estables.

d=Jordi y Montse, **e**=Kepa y Judith, **f**=Antonio y Eva

En este caso, todos los miembros están con su pareja favorita, excepto Antonio que está con su segunda opción, pero él tampoco romperá esta relación porque no puede aspirar a Judith, quien prefiere a Kepa, con quien ya está emparejada.

Lo que consiguieron D. Gale y L. Shapley fue desarrollar un algoritmo con el que siempre se establecen esas parejas estables.

2.1.3 Algoritmo [1]

Condiciones:

Disponemos de dos grupos de elementos, X e Y, con el mismo número de elementos.

Cada elemento tiene una lista ordenada con los elementos del otro grupo.

Objetivo: Formar pares (X,Y) que contengan todos los elementos, de forma que ningún elemento *e* tenga en su lista a un elemento *e´* del otro grupo antes que a su pareja, y que, simultáneamente, *e´* tenga en su propia lista también al elemento *e* antes que a su pareja.

En otras palabras, que ningún elemento prefiera a otro del grupo contrario antes que a su pareja actual y que ese otro elemento no le prefiera a él a su vez.

El procedimiento es sencillo:

Se forma una lista con todos los elementos X, serán los electores. Esta lista representa todos los elementos X libres, temporalmente sin pareja, *XlibresList*

El primer elemento de *XlibresList* sale de la lista y se empareja con su Y favorito.

A continuación el siguiente elemento de *XlibresList,* x_i abandona esta lista y trata de emparejarse con su Y favorito, y_{fav}.

Si y_{fav} no tiene pareja, forman una y se pasa al siguiente X.

Si, en cambio, tiene pareja, $x_{yparejaActual}$, comprueba en su lista si prefiere a x_i o a su pareja actual. En el primer caso, se empareja con x_i y $x_{yparejaActual}$ se añade a la lista de elementos libres *XlibresList.* En el segundo, no hace nada y x_i trata de emparejarse con su siguiente Y favorito.

El procedimiento se repite hasta que XlibresList queda vacío.

2.2 Entornos y software utilizados

Una parte del trabajo, como se ha indicado anteriormente, consiste en la creación de un software que pueda generar elementos de dos grupos diferentes, con características propias cada uno de ellos y capaces de comunicarse entre sí para transmitírselas. A partir de ellas, cada elemento debe formar su lista de favoritos del grupo contrario y cuando todos ellos las han completado ser emparejados con el algoritmo de Gale-Shapley. Estos emparejamientos servirán a un clasificador para entrenarse y tratar de predecir a su vez, con otros conjuntos de parejas si serán parejas estables o no.

Para desarrollar ese software se ha elegido Java como lenguaje de programación porque es con el que el autor está más familiarizado. Por la misma razón se ha elegido Eclipse como entorno de programación.

Aunque en rigor para los objetivos básicos del trabajo no es imprescindible utilizar un sistema multiagente como JADE, ya que los diferentes elementos y las comunicaciones entre ellas se podrían haber implementado igualmente en la forma de instancias y métodos, se ha decidido implementar el sofware desarrollado mediante agentes por varios motivos.

En primer lugar, conceptualmente parece más adecuado que los elementos se comuniquen efectivamente entre sí, que se envíen mensajes y que tengan cierta autonomía ya que se asemeja más al mundo real. Además, aunque en este estudio la ejecución se realiza en una sola máquina, un SMA permitiría que grupos de elementos estuvieran en diferentes máquinas reduciendo la carga en cada de una de ellas y permitiendo el estudio de números de parejas mucho más altos. Por último, como ya se señaló, el autor desea profundizar sus conocimientos en entornos multiagente.

Para finalizar, se ha elegido Weka como sistema de clasificación por estar implementado en Java lo que facilita su integración con el resto de elementos.

2.2.1 Eclipse [10]

Como lenguaje de programación se ha elegido Java y como entorno de programación se ha utilizado Eclipse IDE for Java Developers, software libre de desarrollo de aplicaciones Java muy conocido. Version: 3.8.1.

La descarga de la última versión se puede realizar desde la siguiente página:

https://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/oxygen1

En el Anexo 1 Manual de uso se detalla cómo importar y utilizar el software desarrollado en Eclipse.

2.2.2 JADE [7]

JADE, JavaAgent Develpment Environment, framework que simplifica la implementación de agentes según las especificaciones de la *Foundation for Intelligent Physical Agents* (FIPA). En este apartado se tratará brevemente una primera aproximación al mismo, para más detalles se puede consultar la bibliografía.

Aunque no hay una descripción universalmente aceptada, se puede definir un agente de software como una aplicación informática con capacidad para decidir cómo actuar para conseguir unos objetivos y tiene [12]:

-Autonomía: puede actuar sin intervención directa del usuario
-Capacidad de reacción: percibe el entorno y puede reaccionar a él, por ejemplo, respondiendo a otros agentes con acciones
-Iniciativa: puede actuar por sí mismo, para conseguir sus objetivos, sin necesariamente ser una respuesta a un estímulo externo.

JADE nos permite de forma relativamente sencilla crear agentes, y establecer comunicaciones entre ellos. Brevemente, consiste en:

- Un entorno de ejecución donde los agentes pueden "vivir"
- Una biblioteca de clases que los desarrolladores pueden utilizar para desarrollar sus agentes
- Una serie de herramientas gráficas que facilitan administrar y monitorizar la actividad de los agentes.
-

Cada instancia de un entorno de ejecución constituye un contenedor que puede tener varios agentes. Los contenedores funcionan en una determinada plataforma constituida por una o varias máquinas en red. En cada plataforma, como mínimo existirá un contenedor especial , el primero que se crea, *main container.* El resto de contenedores deben registrarse en él cuando se inician.

Cada contenedor principal, contiene también dos agentes especiales:

Domain Facilitator(DF): Facilitador de dominios, que registrará los servicios que ofrecerán los diferentes agentes.

Agent Management System (AMS): Registra las direcciones de todos los agentes dados de alta.

Utilizando el primero, los agentes pueden encontrar los servicios que necesitan de otros agentes. Gracias al registro de direcciones los agentes pueden enviar mensajes unos a otros.

Cuando el agente se inicia ejecuta una función *setup()* que determinará las primeras acciones que realizará.

Cada agente implementará una serie de comportamientos (*behaviours*) que le indican cómo debe actuar en cada situación. Estos comportamientos se ejecutan de forma concurrente, aunque el programador puede hacer que se ejecuten secuencialmente, una sola vez, cíclicamente, etc...según diferentes tipos ya predefinidos de comportamientos.

La comunicación entre agentes se realiza siguiendo la especificación FIPA-ACL Cada mensaje pertenecerá a la clase *ACLMessage*, del paquete *jade.lang.acl.ACLMessage.* Los mensajes tienen una serie de campos que definen el tipo de mensaje (*perfomative*), el protocolo utilizado(*Protocol*), el emisor del mensaje(*Sender)*, el receptor(*Receiver),* el contenido del mensaje (*Content)* y otros.

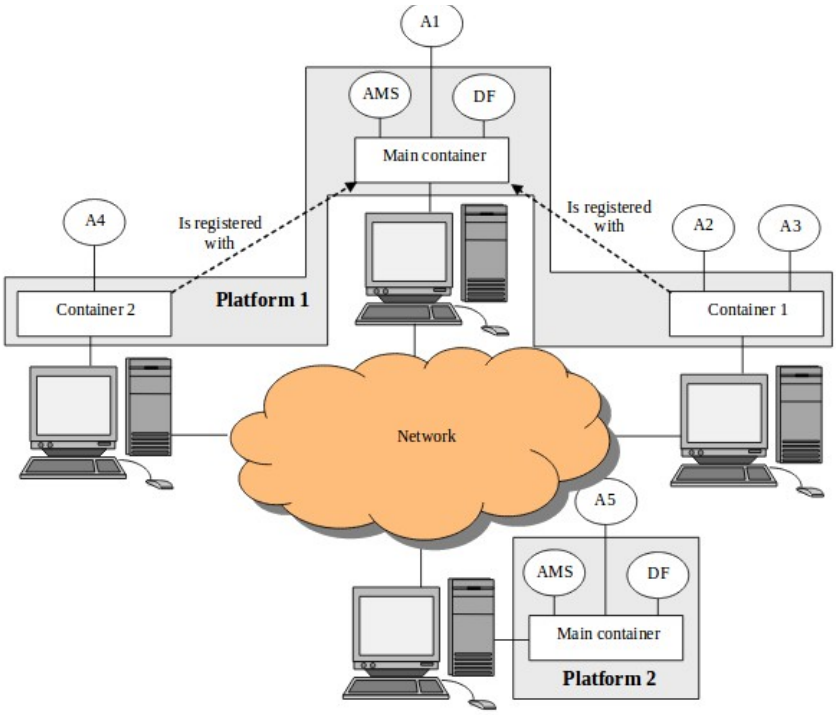

Fig. 2: Arquitectura JADE

Asimismo, JADE incorpora comportamientos predefinidos que facilitan el envío de un mensaje y gestionar las respuestas de otros agentes al mismo. Entre otras, las clases *AchieveREInitiator y AchiveREReponder.*

Además, los mensajes pueden estructurar el contenido de su información mediante una ontología, lo que facilita la comprensión humana de los mismos y la interacción con otros agentes que utilicen la misma ontología.

Todo ello proporciona una forma muy eficaz y potente de transmitir información entre diferentes agentes.

2.2.3 Weka [9]

Weka es un software libre desarrollado por el grupo de Aprendizaje Computacional de la Universidad de Waikato, Nueva Zelanda, para realizar tareas de minería de datos.

El usuario puede estudiar los archivos de datos directamente desde Weka, o utilizar las bibliotecas de algoritmos que incorpora en su propio código Java.

Weka puede leer archivos de datos en su formato propio ARFF (Attribute-Relation File Format), y también en otros tipos de archivos como CSV(*comma-separated values)* que será el que se utilizará en este proyecto.

Los archivos CSV son legibles por el ser humano y representan datos en forma de tabla. Son filas de texto, con elementos separados por comas. Weka interpreta la primera fila como los nombres de unos atributos, cada fila como una instancia y cada elemento limitado por comas como el valor del atributo correspondiente. Se puede ver en el siguiente ejemplo:

Nombre, Edad,Profesión
Antonio, 30, Electricista
Luis, 45, Administrativo
Juan, 37, Médico

Nombre, Edad y Profesión son los nombres de los atributos y cada fila representa una instancia del conjunto de datos. En la primera fila, Antonio sería el nombre de la instancia, 30 su edad y su profesión, electricista.

Weka permite leer esos datos, preprocesarlos con filtros de diversos tipos, desde eliminar atributos a seleccionar determinados rangos de valores. En este trabajo se utilizarán dos de ellos, eliminar un atributo y normalizar los valores.

Asimismo, dispone, entre otros, de un conjunto muy extenso de algoritmos de clasificación y de agrupación.

2.2.4 LibreOffice Calc [13]

Para el análisis de los datos obtenidos en archivos CSV se ha empleado el programa de hoja de cálculo de LibreOffice, Calc, que permite filtrar los rangos de datos con facilidad y generar gráficos.

2.3 Implementación

En este apartado se analizan el diseño y las decisiones más importantes o relevantes tomadas respecto al código del producto.

2.3.1 Descripción del funcionamiento del programa (paquete tfgMarriageStable)

Para ejecutar el programa ver Anexo1: Manual de uso -Manual-tfgMarriageStable.pdf

2.3.1.1 Agentes principales

El programa hace uso de cinco tipos de agentes principales: *GestorAgentes*, *ElementoX*, ElementoY, *Emparejador* y un *Clasificador(*en concreto de su subclase *Weka)*, cada uno de ellos con responsabilidades muy diferentes.

- El ***GestorAgentes*** recibe los parámetros de ejecución del usuario, principalmente el número de parejas que se quieren crear (N) y cuántas características (M) tendrá cada elemento.

  Crea N agentes ElementosX y también N agentes ElementoY asegurando así que habrá el mismo número de ambos grupos. Asimismo crea un agente Emparejador y un agente Clasificador.

- Los diferentes agentes de los tipos **ElementoX y ElementoY** son los elementos que se pretende emparejar. Cada uno de ellos nace con M características (caracteristica0,caracteristica1,caracteristica2…..caracteristicaM).

  Cada característica tiene un valor numérico generado aleatoriamente por el propio agente. Además, por cada característica tiene una preferencia determinada

  (caracteristica0Preferencia,……..……,caracteristicaMPreferencia).

  También cada una de ellas tiene un valor numérico aleatorio.

  Las características tienen valores entre 0 y 1; las preferencias entre 0 y 10

  Cada agente de un grupo pregunta a todos los agentes del otro grupo cuáles son sus características y utilizando sus propias preferencias por cada una de ellas calcula una valoración para cada uno de ellos. Según la valoración de los agentes del grupo contrario los ordena en una lista de favoritos.

  Cuando ha completado y ordenado su lista de favoritos busca los agentes emparejadores y se la envía junto con sus características y preferencias.

- El agente ***Emparejador*** se encarga de encontrar parejas estables.

  Inicialmente se proyectó que esta tarea la realizara el propio gestor de agentes, pero más tarde se decidió asignarla a un agente propio. De esta manera se gana flexibilidad ya que facilita utilizar diversos algoritmos de

emparejamiento, e incluso se podrían llegar a emplear varios agentes emparejadores simultáneamente.

En este proyecto el emparejador utiliza únicamente el algoritmo de Gale-Shapley.

Cuando ha terminado de ejecutar el algoritmo guarda los resultados en un archivo CSV (*comma-separated values*). Cada línea representa una pareja y guarda las características y preferencias de cada miembro de la pareja y si es estable o no estable.

Por último, envía un mensaje al agente *Clasificador* comunicándole la ruta y el nombre de ese archivo para que pueda analizarlo.

- El agente ***Clasificador(subclase Weka)*** si es creado por el gestor de agentes queda a la espera de recibir mensajes del *Emparejador*. No obstante, también puede ser creado directamente por el usuario y pasarle como argumentos la ruta y nombre de uno o dos archivos de datos[1]. El primero de estos archivos servirá de conjunto de entrenamiento y el segundo de conjunto de test.

  Una vez que el ***Clasificador***, obtiene los archivos para analizar, bien como contenido de un mensaje de un *Emparejador,* bien como argumentos iniciales, carga los datos, los preprocesa, eliminando el primer atributo (nombre de los elementos que forman la pareja), normalizando los datos y procede a analizarlos con un algoritmo de clasificación Weka, en este caso Logistics[14]

  Si solo se ha introducido un archivo procederá a efectuar una validación cruzada de diez iteraciones[15]. Si se han introducido dos archivos, el primero actuará como conjunto de entrenamiento y el segundo como conjunto de test.

Una vez realizado el análisis se muestra su resultado en pantalla

1 Consultar Anexo1 Manual de usuario6.1.3.3.1 Opciones Avanzadas (añadir argumentos)/Clasificador

```
=== Confusion Matrix ===

     a     b   <-- classified as
    29   271 |     a = Estable
    30 10124 |     b = NoEstable

El evaluador ha tardado: 2771 milisegundos
Archivo de entrenamiento:GaleShapleyCompleto_parejas_100_Iteraciones_3_2
Archivo de test:null
Aciertos:10153.0
Errores:301.0
Precision:97.12%
Porcentaje error:2.88%
Tasa true positives Estables:0.10
Tasa true positives No Estables:1.00
Precision Estables:0.49
Precision No Estables:0.97
F-Measure Estables:0.16
F-Measure NoEstables:0.99
Kappa estadístico:0.15
```

Fig. 3: Resultado de una clasificación

Asimismo guarda en el archivo *resultados.csv* los parámetros más importantes de la clasificación. Más adelante, en la sección de estudio de resultados, se explicarán cuáles son y el uso que se les da en el trabajo.

2.3.1.2 Opciones de ejecución

El programa permite añadir algunos parámetros adicionales a la ejecución de GestorAgentes de manera que modifiquemos su funcionamiento[2].

Argumentos admitidos al crear un **gestor de agentes**:

“fp”	Guarda en dos archivos (elementosX.out y elementosY.out) las listas de preferencias de los agentes de cada grupo. Si los archivos existen los sobrescribe. Por defecto no se modifican. Servirán para comprobar de forma solitaria el algoritmo de GaleShapley.
Un número entero: n	Para generar un conjunto mayor de datos de estudio del emparejamiento de k-parejas se pueden realizar **iteraciones**, repitiendo n-veces la creación de elementos y sus emparejamientos. Importante: Esto no implica que crear 20 parejas con 5 iteraciones sea igual a crear y emparejar 100 parejas. Lo único que se hace es

2 Consultar Anexo1 Manual de usuario 6.1.3.3.1 Opciones Avanzadas (añadir argumentos)/GestorAgentes

	repetir el emparejado de 20 parejas 5 veces para tener un conjunto de datos 5 veces mayor del emparejado de 20 parejas. Opción por defecto: Una sola iteración.
“completo”	Por defecto, el emparejado guarda solo una parte de las parejas no estables que se encuentran. Con esta opción se guardan todas ellas. Para ver la diferencia podemos considerar que al generar 100 parejas en una determinada ejecución se han producido 410 parejas no estables (no completo) y con la opción completo 3524 parejas no estables.
“moduladas”	Por defecto las características y preferencias de un elemento, X o Y, son totalmente aleatorias. Activando esta opción los agentes X darán más valor a unas características específicas, mientras que los agentes Y a otras.
“newfile”	Guarda en un nuevo archivo el resultado del emparejamiento en lugar de sobrescribir el archivo Datos.csv como hace por defecto. Este archivo adopta la forma: GaleShapleyCompleto_parejas_100_Iteraciones_1_2017_12_13_10_07_24modulacion:false.csv Nos indica que se ha activado la opción completo, se han generado 100 parejas en una sola iteración, que el archivo se ha creado el 13/12/2017 a las 10:07:24 y que no se ha activado la modulación.

Merece la pena examinar con más detalle algunos de ellos.

- Número de iteraciones: Durante el desarrollo del proyecto se advirtió que para números de parejas bajos el número de instancias que se pasarían al clasificador para analizar podría ser excesivamente bajo.

Para corregir esto se desarrolló un sistema con el que al iniciar el Gestor de Agentes se le podía indicar que repitiera el proceso un número de veces determinado para obtener más volumen de datos para analizar

Así cuando el emparejador termina de asociar las parejas estables, envía un mensaje al gestor de agentes pidiéndole que continúe el programa.

El gestor de agentes, si está en la última iteración, envía un *Agree* al Emparejador quien crea el archivo de datos y lo envía al Clasificador. Pero si no está en la última iteración, responde un *Refuse* y deja al Emparejador a la espera. Después, el Gestor de Agentes destruye todos los ElementosX y ElementosY y crea una nueva generación repitiéndose el ciclo hasta que se han completado todas las iteraciones.

- Completo: El algoritmo de Gale-Shapley, como hemos visto, se ejecuta construyendo parejas provisionales que se rompen cuando se encuentra una pareja mejor.

Esta opción permite al usuario distinguir entre guardar en el conjunto de datos de entrenamiento como no estables solo las parejas parejas que efectivamente se rompen (modo normal); o también las parejas que no llegan a formarse porque los elementos prefieren a sus parejas provisionales antes que a un nuevo pretendiente.

Esto, que puede parecer trivial, tiene gran importancia en el volumen de datos ya que con números de parejas grandes varía mucho. Un ejemplo: en una ejecución normal, de 100 parejas, se han obtenido 410 parejas no estables, pero activando el modo completo se han obtenido 3524, ocho veces y media más. Con números de parejas mayores la diferencia aumenta todavía más.

Este hecho plantea la cuestión de si una ejecución completa dará demasiado peso a las instancias de clase no estable muchísimo más numerosas que las no estables. La opción de utilizar una u otra ejecución permitirá estudiar su impacto en la clasificación.

La implementación de esta opción se ha realizado creando dos clases hijas de Emparejador, convirtiendo este en una clase abstracta. La clase *GaleShapley* realiza el emparejamiento incompleto mientras que *GaleShapleyCompleto* guarda todas las parejas no estables. El Gestor de Agentes creará una u otra según los parámetros introducidos por el usuario.

Este diseño, basado en la herencia de clases, tiene la ventaja de que se pueden añadir fácilmente nuevos algoritmos de emparejamiento en nuevas clases hijas de Emparejador.

- Moduladas: Como se ha indicado, cuando los agentes *ElementoX* y *ElementoY* se crean sus características y preferencias son totalmente aleatorias y pertenecen a los mismos rangos de valores en ambos tipos.

Sin embargo, en la vida real, puede haber con frecuencia situaciones en las que unas características sean mucho más importantes que otras para valorar a un elemento de otro grupo. Es concebible que esto tenga un impacto en la clasificación.

Para poder examinar este fenómeno se ha añadido esta opción que, si se activa, modifica cómo se generan las preferencias de los elementos X e Y. Los elementos X darán más importancia a las primeras características y menos a las últimas de manera progresiva, mientras que los elementos Y, además, lo harán a la inversa.

2.3.1.3 Clases

Dada la complejidad del programa y la gran cantidad de métodos y comportamientos utilizados, se dará de cada clase una breve explicación. La explicación más exhaustiva puede consultarse en la carpeta doc del proyecto así como en los comentarios del código.

GestorAgentes (extiende agent)

Al iniciarse ejecuta, como cualquier agente, la función setup().

Abre un panel de diálogo con el usuario y pregunta cuántas parejas se quieren crear y después cuántas características ha de tener cada elemento. De esta forma, se podrá realizar el estudio con diferentes números de parejas y también con elementos con distintas características.

Recoge los argumentos de las opciones introducidos por el usuario[3]

Si se ha activado la opción “completo” crea un *Emparejador GaleShapleyCompleto*, en caso contrario crea un *Emparejador GaleShapley*.

Crea un Clasificador *Weka*.

A continuación crea tantos pares de *Elementos X* y *Elementos Y* como número de parejas haya indicado el usuario.

Una vez ha creado todos los agentes, busca los elementos de los grupos X e Y. Si no es capaz de encontrarlos tras 50 intentos, obteniendo siempre un mismo resultado insuficiente, muestra un mensaje de error y cierra el programa. Esto es una medida de protección para evitar que el sistema quede indefinidamente buscando parejas.

Si la búsqueda es satisfactoria envía cada uno de ellos un mensaje *ACL* con la lista de los AID de los elementos del grupo con quien se pueden emparejar.

Asimismo, crea un manejador de respuestas para los *Request* de los Emparejadores

```
class ManejadorResponderRequests extends AchieveREResponder
```

Se ha señalado ya que cuando un Emparejador termina un emparejamiento envía un mensaje al *Gestor de Agentes*. Ese mensaje es un *Request* con el contenido “*continueProgram*”. El manejador del Gestor de Agentes comprueba si se trata de la última iteración o no. Si es la última iteración responde con un *Agree* y simplemente destruye los agentes X e Y que ya no harán falta. En cambio, si no es la última iteración responde con un *Refuse,* destruye los agentes, y crea una nueva generación de ellos repitiendo el ciclo inicial hasta que se completan todas las iteraciones.

Adicionalmente, el Gestor de Agentes también se ocupa de registrar el tiempo que tardan los elementos X e Y en comunicarse entre sí y completar todas las listas de favoritos. Para ello abre el siguiente comportamiento:

```
EscucharConversationId extends Behaviour
```

3 Ver 2.3.1.2 Opciones de Ejecución

Este comportamiento permite al agente escuchar mensajes con un determinado identificador de conversación. En particular, escucha los mensajes con identificador de conversación “*tiempoInicioConsultas*”.

Los diferentes Elementos envían mensajes al gestor de agentes con ese identificador en el momento en que han completado y ordenado su lista de favoritos. Cuando recibe el primer mensaje de este tipo, el Gestor de Agentes inicia un contador de tiempo y cuando ha recibido un número de mensajes igual al número de parejas*2 significa que todos los agentes han completado sus listas por lo que finaliza el contador de tiempo.

Registra estos tiempos en el archivo *comunicaciones.csv*

Elemento (extiende agent)

Representa cada uno de los elementos (agentes) que hay que emparejar. Se trata de una clase abstracta ya que cualquier elemento ha de ser de la clase *ElementoX* o E*lementoY.*

Cuando se inicia el Elemento, recibe como argumentos el AID del gestor de agentes que lo ha creado, el número de parejas, el número de características, el número de iteración en curso, y si hay modulación o no.

Con el número de características crea dos *ArrayList*, uno para guardar sus propias características y otro para guardar sus preferencias respecto a esas características.

Cada característica, o preferencia, está formada por un nombre (*String*) y un valor (*float*) y está representada por la clase *Caracteristica.*

El *Elemento* llama a dos funciones: *iniciarCaracteristicas()* e *iniciarPreferencias(),* que automáticamente asignan esos nombres y unos valores de manera aleatoria.

A continuación el *Elemento* se registra en el *Domain Facilitator.* Si es de la clase *ElementoX* lo hará ofreciendo el servicio “*conocerParejasY*” y si es de la clase *ElementoY* lo hará ofreciendo el servicio “*conocerParejasX*”. Este registro permitirá al Gestor de Agentes encontrarlos cuando los busque.

Inmediatamente después inicia el comportamiento:

```
EscucharObjetoConversation extends Behaviour
```

Este comportamiento le permite recibir mensajes con un determinado identificador de conversación, en concreto se utiliza “listaParejas”. El Gestor de Agentes envía la lista de elementos del grupo con ese identificador así que el Elemento puede recibirla y crea con ella una lista de potenciales parejas a las que asigna una valoración inicial de 0.

A continuación, envía a cada uno de ellos una *Query* preguntando por sus características.

El comportamiento `class ResponderQuery extends AchieveREResponder`

permite al Elemento responder esas *Queries* y envía como respuesta también como objeto serializado un ArrayList con sus características.

De esta forma cada elemento envía peticiones de información de sus características a otros agentes y a su vez responde a las peticiones de otros agentes enviándoles las suyas.

Cuando se recibe una respuesta de otro agente, si todo ha ido bien, debería ser un ArrayList con las características del emisor de la misma. Entonces el *Elemento* calcula con sus preferencias una valoración para el agente que la ha enviado. Lo hace multiplicando las características por sus propias preferencias para cada característica y sumándolas. Para ello utiliza la función calcularValoracion().

Ejemplo:

Características recibidas: [caracteristica0=0.3, caracteristica1=0.4]
Preferencias del agente: [caracteristica0Preferencia=2, caracteristica1Preferencia=5]

Valoracion=0.3*2+0.4*5=2.6

El siguiente paso es actualizar la valoración del emisor de la respuesta en la lista de parejas y se ordena esta de mayor a menor valoración. Cuando el número de valoraciones realizadas por el agente iguala el número de parejas se crea una instancia de *ElementoCompleto* que representa el elemento con su AID, los AID de su lista de parejas ordenados de mayor a menor valoración y sus propias características y preferencias.

Finalmente, envía esa información al emparejador en un mensaje cuyo contenido es el *ElementoCompleto* que ha creado.

ElementoX y ElementoY (extienden Elemento)

Aunque la mayoría de las propiedades y métodos de los elementos X e Y son comunes y están recogidas en la clase madre Elemento, se han diferenciado ambos en dos clases separadas.

En primer lugar porque son elementos de grupos disjuntos; en segundo, porque de esta forma podemos utilizar métodos propios en cada uno de ellos. En particular, cuando se ha activado la opción “modulación” los elementos X e Y generarán sus preferencias de forma diferente.

Si existen M características, los elementos X multiplicarán el número aleatorio generado para cada preferencia por un número que dependerá del número de la característica, de esta forma:

caracteristica0Preferencia=aleatorio*M
caracteristica1Preferencia=aleatorio*(M-1)
.
.
.
caracteristica(M-1)Preferencia=aleatorio*1

Los elementos Y lo harán exactamente a la inversa

caracteristica0Preferencia=aleatorio*1
caracteristica1Preferencia=aleatorio*2
.
.
.
caracteristica(M-1)Preferencia=aleatorio*(M-1)

Los elementos X otorgan un mayor valor a las primeras características, mientras que los elementos Y lo hacen con las últimas.

Caracteristica (implementa Serializable)

Esta clase representa una característica, o una preferencia, de un *Elemento*. Está formada por el nombre de la característica (*String*) y un valor (*float)*

Implementa *Serializable* para poder ser enviada como contenido de un mensaje.

ElementoValorado (implements Comparable<ElementoValorado>)

La utilidad de esta clase es que permite a los Elementos comparar los elementos del otro grupo y gracias a eso ordenarlos en la lista de favoritos.
Guarda el AID del elemento y una valoración para él.

El método *compareTo* permite comparar dos ElementoValorado según sus valoraciones. Aquél que tenga una valoración más alta será “mayor que” otro elemento que tenga una valoración más baja.

Por su parte, el método *updateValoracion* posibilita cambiar la valoración de un ElementoValorado.

ListaParejas (extiende ArrayList)

Representa las parejas de un *Elemento.* Dado que cada uno de los elementos del arreglo son ElementoValorado, que implementa *Comparable,* la lista puede hacer uso del método *ordenarMayorAMenor* y ordenarse de mayor a menor valoraciones.

ElementoCompleto (Implementa Serializable)

Las instancias de esta clase se construyen a partir del nombre local de un Elemento, su lista de características, su lista de preferencias, su lista de parejas y la clase a la que pertenece.

Contiene toda la información de un *Elemento* que necesita el *Emparejador* para realizar su tarea.

Pareja

Clase que representa una pareja Tiene dos atributos, *strings*, cada uno de ellos es el nombre de un miembro de la parejas

Emparejador (extiende Agent)

Agente encargado de realizar el emparejamiento estable de los elementos. Cuando se inicia crea varios mapas clave valor:

- **`elementosX`**

Elementos X recibidos en una iteración, la clave del mapa es el nombre del elemento y el valor su lista de favoritos.

- **ElementosY**

Elementos Y recibidos en una iteración, la clave del mapa es el nombre del elemento y el valor su lista de favoritos.

- **elementosCompletos**

Guarda todos los *ElementoCompleto* recibidos en todas las iteraciones. La clave del mapa es el nombre de un elemento y el valor el *ElementoCompleto* correspondiente.

- **Emparejados**

Guarda las parejas estables de cada iteración, la clave del mapa es el nombre de un elemento Y y el valor el nombre del elemento X.

- **EmparejadosFinal**

Guarda las parejas estables de todas las iteraciones, la clave del mapa es el nombre de un elemento Y y el valor el nombre del elemento X.

Asimismo crea dos ArrayList en los que guardará las parejas no estables.

noEmparejados guarda las parejas no estables de una iteración mientras que **noEmparejadosFinal** las de todas las iteraciones.

Cuando se inicia este agente recibe como argumentos el número de parejas, el AID del Gestor de Agentes, si hay que guardar un archivo de pruebas (*filePruebas*), si hay que guardar los resultados en un archivo nuevo (*newfile*) y si las características están moduladas o no (*moduladas*)[4]

Inmediatamente, se registra ofreciendo el servicio de *name*="emparejar" *type*="parejas" *owner*="TFG". Esto permite que los diferentes Elementos X e Y le encuentren y puedan enviarle mensajes.

A continuación, crea inicia un comportamiento manejador:

```
class ManejadorResponderRequests extends AchieveREResponder
```

Con este manejador escucha los *Request* de los elementos que le piden un emparejado y recibe los ElementoCompleto de cada uno de ellos. Es decir, por cada Elemento recibe su nombre local, su lista de características, su lista de preferencias, su lista de favoritos ordenada de mayor a menor y la clase a la que pertenece, X o Y.

El Emparejador guarda cada *ElementoCompleto* recibido en el mapa elementosCompletos

4 Entre paréntesis los nombres de las variables que guardan la información.

También guarda en *elementosX y elementosY* los elementos recibidos y sus listas de favoritos.

Cuando ha recibido todos los elementos llama al comportamiento

```
class Emparejar extends OneShotBehaviour
```

Este comportamiento llama al método Emparejar() el cual utiliza los mapas *elementosX, emparejadosY y el ArrayList noEmparejados* para formar parejas estables que guardará en el mapa *emparejados;* y no estables que guardará en *noEmparejados.*

Es importante aquí señalar que el método Emparejar() se sobrescribe en las clases *GaleShapley* y *GaleShapleyCompleto*, que serán quienes realmente utilicen un emparejado u otro.

Después, añade las parejas estables y no estables encontradas a los mapas correspondientes que guardan las parejas de todas las iteraciones emparejadosFinal y noEmparejadosFinal.

Hecho esto, reinicia los mapas de la iteración en curso y envía un mensaje al Gestor de Agentes pidiéndole que continúe el programa y queda a la espera de su respuesta.

Si recibe un *Refuse*, no hace nada y espera los ElementoCompleto de una nueva iteración, pero si recibe un *Agree* significa que han terminado todas las iteraciones, en cuyo caso inicia el comportamiento:

```
class ArchivoResultado extends OneShotBehaviour
```

Este comportamiento guarda en un archivo CSV el resultado del emparejamiento. Por defecto este archivo se llama Datos.csv y se sobrescribe, pero si el usuario ha activado la opción "*newfile*" se crea un nuevo archivo en cuyo nombre se almacena la información del tipo de algoritmo usado, el número de parejas, el número de iteraciones y si ha habido modulación o no.

Ejemplo:
GaleShapleyCompleto_parejas_100_Iteraciones_1_2017_12_13_10_07_24modulacion_false.csv

Indica que se ha activado la opción completo, se han generado 100 parejas en una sola iteración, que el archivo se ha creado el 13/12/2017 a las 10:07:24 y que no se ha activado la modulación.

Cada fila del archivo CSV creado representa una pareja y las características de cada elemento.

La primera columna son los nombres de los elementos que constituyen la pareja, separados por el símbolo ":"

elementoX_48 : elementoY_69

Las siguientes columnas son las características y preferencias del elemento X de la pareja, seguidas de las características y preferencias del elemento Y

Xcaracteristica0 Xpreferencia0 Xcaracteristica1 Xpreferencia1 Ycaracteristica0 Ypreferencia0 Ycaracteristica1 Ypreferencia1

La última columna indica si la pareja es estable o no estable.

Una vez creado el archivo, el Emparejador envía un mensaje al Clasificador cuyo contenido es el nombre y ruta del archivo.

Estos archivos que genera el Emparejador son una de las claves de este estudio ya que son los que relacionan las características de dos elementos con su estado de pareja estable o no estable y los que posteriormente leerá el clasificador Weka

GaleShapley y GaleShapleyCompleto

Como ya se ha dicho, son clases hijas de Emparejador y se utilizan para sobrescribir el método Emparejar(). En el primer caso se guardarán en el *ArrayList noEmparejados* solo las parejas provisionales que una vez formadas se hayan roto, mientras que en el segundo se añadirán también las parejas que no llegan a formarse porque los elemento Y prefieren en un momento dado a su pareja provisional a las de su X pretendiente.

Ejemplo: Supongamos que un elementoX, X se propone a un elementoY ,Y quien está ya emparejado con otro elementoX, Z.

Si Y prefiere a X la pareja que forma con Z ambos elementos añadirán la pareja rota, Y, Z a su lista de noEmparejados.

Pero si Y no rompe su pareja, sino que permanece con Z, GaleShapleyCompleto añade a noEmparejados la pareja que no se llega a formar (Y,X), mientras que GaleShapley no lo hace.

Ambas clases guardan en el archivo *tiemposEmparejamientos.csv* el número de parejas que han emparejado, los milisegundos que han tardado en el proceso, la iteración en que se ha producido y si se ha activado la opción “completo” o no. Este archivo permitirá comparar con los tiempos que emplea el clasificador en realizar una evaluación.

Clasificador

Esta clase se responsabiliza únicamente de registrarse para que los agentes Emparejador puedan encontrarla, recibir el nombre del archivo CSV del emparejador, y llamar al método *UtilizarDatos()* que, en esta clase no está implementado, lo está en su clase hija *Weka*, el verdadero clasificador.

Weka

Como clase hija de Clasificador, Weka recibe el nombre del archivo a analizar, lo lee y lo carga en memoria como un conjunto de instancias. Sin embargo, también puede ser creada directamente por el usuario y recibir uno o dos nombres de archivos

como argumentos[5]. En ese caso cargará esos archivos y utilizará el primero como conjunto de entrenamiento y el segundo como conjunto de prueba.

En cualquiera de ambos casos, aplica a los datos cargados de todos los archivos dos filtros. Eliminará la primera columna (que guarda los nombres de las parejas y son irrelevantes) y normalizará los datos.

Hay que normalizar los datos porque son numéricos pero tienen escalas distintas, las características adoptan valores de 0 a 1 mientras que las preferencias de 0 a 10, y si la modulación está activada serán aun mayores.

Una vez realizada la normalización crea a partir del archivo enviado por el Emparejador (o del del primer argumento introducido por el usuario en una invocación directa del agente Weka) un clasificador *Logistic* [14] y lo evalúa.

Si solo se dispone de un archivo de datos, la evaluación se realiza utilizando una validación cruzada [15] de 10 iteraciones. Eso quiere decir que se divide el conjunto de datos en 10 partes, se reserva una como conjunto de prueba y se utiliza el resto como conjunto de entrenamiento. Se repite 10 veces utilizando cada vez una parte distinta como conjunto de prueba y finalmente calcula los resultados medios de las 10 iteraciones. Utiliza para ello el método *crossValidateModel* de la clase *Evaluation* [16] de las bibliotecas Weka

Si el agente ha recibido dos archivos utiliza el primero como entrenamiento del clasificador *Logistic* y evalúa la clasificación sobre el conjunto de prueba mediante el método *evaluateMode*l de la clase *Evaluation* de las bibliotecas Weka*.*

Finalmente, muestra el resultado de la evaluación[6] y guarda los datos de la misma en el archivo resultados.csv[7]

Funciones

Contiene diversas funciones auxiliares a diferentes clases. Las más importantes son

```
float getDecimal()
```

Obtiene un número entre 0 y 1.

```
float getMultiplicador()
```

Obtiene un número entre 0 y 10.

```
registrarLenguajeSL(Agent pAgent)
```

Registra el lenguaje SL para el agente pAgent.

```
void cerrarSistema(Agent agent)
```

Destruye todos los agentes activos y cierra la plataforma JADE.

```
Map emparejarGaleShapley(Map elementosX, Map elementosY, ArrayList
noEmparejados)
```

Empareja los elementos de dos grupos dadas sus preferencias según el algoritmo de Gale-Shapley. A medida que forma las parejas añade solo las parejas provisionales

5 Ver Anexo 1: Manual de usuario 6.1.3.3.1 Opciones avanzadas (añadir argumentos)
6 Ver Fig.3 Resultado de una clasificación
7 Ver 2.4.2.1 Archivo con resultados

que rompen a noEmparejados. Retorna un mapa con las parejas estables en las que las claves son los nombres de los agentes Y y los valores los nombres de los elementos X.

```
Map emparejarGaleShapleyCompleto(Map elementosX, Map elementosY, ArrayList noEmparejados)
```

Empareja los elementos de dos grupos dadas sus preferencias según el algoritmo de Gale-Shapley. A medida que forma las parejas añade todas las parejas no estables a noEmparejados, tanto las que se rompen, como las que no se llegan a formar porque un elemento Y rechaza a un pretendiente X. Retorna un mapa con las parejas estables en las que las claves son los nombres de los agentes Y y los valores los nombres de los elementos X.

```
void createFile(String pTexto, String nameFile, Boolean continuarEnFichero)
```

Crea un archivo con el nombre nameFile y cuyo contenido es la cadena pTexto. Si continuarEnArchivo es *true* continua escribiendo en el archivo si ya existía, si es *false* lo sobrescribe.

2.3.2 Paquete de pruebas unitarias de Gale-Shapley (paquete pruebasGaleShapley)[8]

Dada la importancia capital en este estudio del algoritmo de Gale-Shapley se ha desarrollado un paquete de pruebas para analizar si su implementación es correcta y que permite comprobar si un emparejamiento está formado por parejas estables o no. Este paquete es totalmente independiente del paquete del programa principal tfgMarriageStable y puede utilizarse por separado.

Para utilizar el paquete, consultar el Anexo 1:
Manual de uso -Manual-tfgMarriageStable.
Sección 6.1.3.2.2 Paquete pruebasGaleShapley

Consta de cuatro clases. Dos de ellas, *GaleShapleyUnitario* y *CheckParejasFiles*, contienen una *Main Class*, por lo que son ejecutables mientras que las otras dos, *Funciones* y *Parejas*, son clases auxiliares.

GaleShapleyUnitario se encarga de formar parejas estables según el algoritmo de Gale-Shapley, mientras que *CheckParejasFiles* comprueba si unas parejas dadas son estables.

Se suministran dos archivos elementosX.out y elementosY.out que contienen ejemplos reales de listados de los elementos de cada grupo junto con sus listas de favoritos:

Los dos primeros contienen elementos de cada grupo y sus listas de favoritos.

8 Todos los ejemplos de este apartado son ejecuciones reales que se han empleado para comprobar la eficacia del algoritmo implementado. También se ha comprobado su funcionamiento con archivos generados para números de parejas más altos.

elementoX_0:elementoY_2,elementoY_3,elementoY_1,elementoY_4,elementoY_0
elementoX_1:elementoY_2,elementoY_3,elementoY_1,elementoY_4,elementoY_0
elementoX_2:elementoY_2,elementoY_3,elementoY_1,elementoY_4,elementoY_0
elementoX_3:elementoY_2,elementoY_3,elementoY_4,elementoY_1,elementoY_0
elementoX_4:elementoY_2,elementoY_3,elementoY_1,elementoY_4,elementoY_0

elementoY_4:elementoX_1,elementoX_2,elementoX_4,elementoX_3,elementoX_0
elementoY_3:elementoX_1,elementoX_2,elementoX_3,elementoX_4,elementoX_0
elementoY_2:elementoX_1,elementoX_3,elementoX_2,elementoX_4,elementoX_0
elementoY_1:elementoX_4,elementoX_1,elementoX_2,elementoX_3,elementoX_0
elementoY_0:elementoX_1,elementoX_4,elementoX_3,elementoX_0,elementoX_2

Según estos archivos, elementoX0 prefería en primer lugar emparejarse con elementoY2, si no puede con este, con elementoY3, si tampoco puede con este le gustaría con el elementoY1, seguido de elementoY4 y solo como última opción aceptaría emparejarse con elementoY0.

El resto de elementos se leerían de manera análoga.

Además, se suministra un archivo, parejas.txt, que contiene una serie de parejas formadas.

elementoX_3:elementoY_4
elementoX_2:elementoY_3
elementoX_1:elementoY_2
elementoX_4:elementoY_1
elementoX_0:elementoY0

El elementoX_3 está emparejado con el elementoY4, X2 con Y3, etc.…

Hay que señalar que el contenido de estos archivos aquí mostrados son el resultado de una ejecución determinada y que el contenido de los elementos suministrados puede variar.

Estos archivos se pueden crear manualmente, pero también se pueden generar automáticamente.

Si en una ejecución del programa principal activamos la opción “fp”, el Emparejador creará, si ya existen sobrescribirá, dos archivos, elementosX.out y elementosY.out, con las listas de preferencias de los elementos generados.

En cuanto al archivo parejas.txt, se sobrescribirá, como se verá, cada vez que ejecutemos el programa *GaleShapleyUnitario*.

GaleShapleyUnitario

Lee el contenido de los archivos elementosX.out y elementosY.out y los guarda en sendos mapas, *elementosX*, *elementosY* en los que las claves son los nombres de los elementos y los valores sus listas de favoritos.

Crea un ArrayList que guardará las parejas no estables, noEmparejados

Invoca la función emparejarGaleShapleyCompleto(elementosX, elementosY, noEmparejados) y empareja de forma estable siguiendo el algoritmo de GaleShapley, en su modo completo, los elementos de ambos grupos.

Muestra en pantalla las listas de preferencias de cada elemento y las parejas tanto estables como no estables.

```
.....................................

   LISTAS DE PREFERENCIAS DE LAS PAREJAS
.....................................

elementoX_0: [elementoY_3, elementoY_0, elementoY_2, elementoY_4, elementoY_1]
elementoX_1: [elementoY_1, elementoY_3, elementoY_4, elementoY_0, elementoY_2]
elementoX_2: [elementoY_3, elementoY_0, elementoY_4, elementoY_2, elementoY_1]
elementoX_3: [elementoY_0, elementoY_3, elementoY_1, elementoY_2, elementoY_4]
elementoX_4: [elementoY_3, elementoY_0, elementoY_4, elementoY_2, elementoY_1]

....................

elementoY_4: [elementoX_3, elementoX_0, elementoX_4, elementoX_2, elementoX_1]
elementoY_3: [elementoX_3, elementoX_1, elementoX_0, elementoX_4, elementoX_2]
elementoY_2: [elementoX_3, elementoX_0, elementoX_4, elementoX_2, elementoX_1]
elementoY_1: [elementoX_3, elementoX_0, elementoX_4, elementoX_2, elementoX_1]
elementoY_0: [elementoX_3, elementoX_2, elementoX_0, elementoX_4, elementoX_1]


PAREJAS ESTABLES:

elementoX_4,elementoY_4
elementoX_0,elementoY_3
elementoX_2,elementoY_2
elementoX_1,elementoY_1
elementoX_3,elementoY_0


 PAREJAS: 5


PAREJAS NO ESTABLES:

elementoX_2,elementoY_3
elementoX_2,elementoY_0
elementoX_4,elementoY_3
elementoX_4,elementoY_0
elementoX_2,elementoY_4


 PAREJAS: 5
```

Además guarda las parejas estables encontradas en el archivo parejas.txt sobrescribiéndolo.

CheckParejasFiles

Lee los archivos parejas.txt, elementosX.out y elementosY.out.

Guarda el contenido de parejas.txt en un mapa, *emparejados*, cuyas claves serán los nombres de los elementos Y de cada pareja y los valores, el elemento X de cada pareja.

Guarda también en sendos mapas el contenido de elementosX.out y elementosY.out, de forma que en cada mapa tenemos como claves los nombres de un elemento y como contenidos sus listas de favoritos.

Recorre el mapa formado con las parejas leídas del archivo parejas.txt.

Para cada clave, elementoY examina su valor, elementoX y busca en el mapa elementosX su lista de favoritos.

Busca en esa lista la posición del elemento Y

Después solo hay que recorrer toda la lista de favoritos de X y comparar las posiciones en que se encuentra X para cada uno de ellos, con la posición de la pareja con la que se encuentran. Si se encuentra un elemento Y que tenga en su lista antes a X que a su pareja actual y no se ha alcanzado todavía la posición de la pareja actual de Y la pareja es no estable y se mostrará un mensaje de error.

Si el algoritmo de GaleShapley aplicado al crear el archivo parejas.txt es correcto no debería producirse ningún error. Sin embargo, para comprobar el funcionamiento el usuario puede editar el los archivos out, modificando las listas de preferencias o, más fácil aún, creando una pareja no estable en parejas.txt, utilizando la información obtenida al ejecutar GaleShapleyUnitario.

Ejemplo: Sabiendo que con las preferencias dadas las siguientes parejas son no estables

```
elementoX_2,elementoY_3
elementoX_2,elementoY_0
elementoX_4,elementoY_3
elementoX_4,elementoY_0
elementoX_2,elementoY_4
```

Si se modifica el archivo de parejas.txt de la siguiente forma:

parejas.txt original

```
elementoX_4,elementoY_4
elementoX_0,elementoY_3
elementoX_2,elementoY_2
elementoX_1,elementoY_1
elementoX_3,elementoY_0
```

parejas.txt modificado

```
elementoX_4,elementoY_4
elementoX_0,elementoY_2
elementoX_2,elementoY_3
elementoX_1,elementoY_1
elementoX_3,elementoY_0
```

Al ejecutar GaleShapleyUnitario, se debería obtener al menos un error. Y, efectivamente, así es, siendo estos los mensajes de error que proporciona el sistema.

```
ERROR: elementoY_3 y elementoX_4 preferirían estar juntos antes que con quienes están elementoX_2, elementoY_4

ERROR: elementoY_3 y elementoX_0 preferirían estar juntos antes que con quienes están elementoX_2, elementoY_2
```

Funciones

Empaqueta las funciones auxiliares que utilizarán los dos programas. Lo más relevante respecto a nuestro estudio es que el código de la función que realiza el emparejado de Gale-Shapley es exactamente la misma que lo realiza en el programa principal. Solo se diferencian en que los mapas y arreglos de cada función

son interpretados de forma distinta según en qué paquete se encuentren. En el paquete principal se utilizan los del paquete *jade.util.leap*, mientras que en el del paquete de pruebas se utilizan los del paquete *java.util*.

Pero esta diferencia se produce en tiempo de compilación, el código en ambos casos es idéntico.

Pareja

Clase auxiliar que representa una pareja Tiene dos atributos, *strings*, cada uno de ellos es el nombre de un miembro de la pareja.

2.3.3 Diagrama de clases

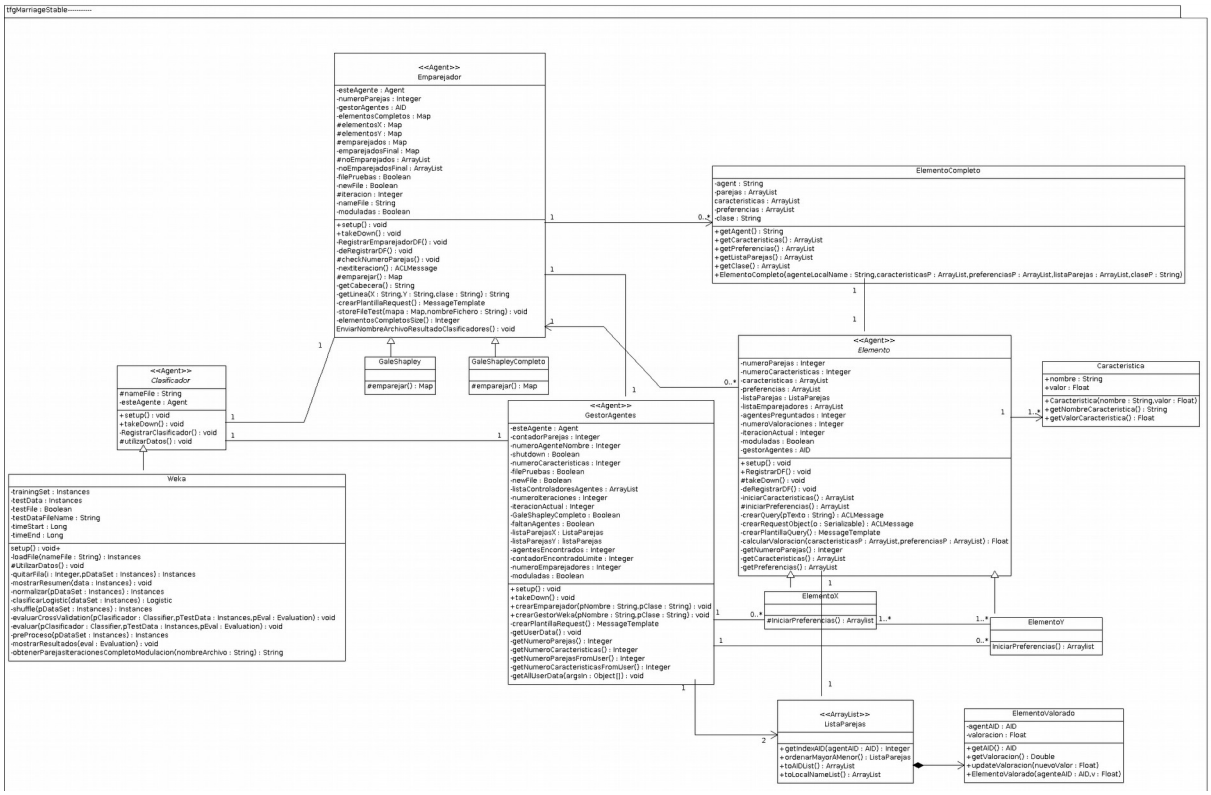

Fig. 4: Agentes y clases auxiliares

Se muestran en primer lugar, en la figura 4, los diferentes agentes y algunas clases auxiliares. Por no sobrecargar el diagrama y simplificarlo se ha indicado entre <<>>[9] las clases de otros paquetes, como JADE, de las que la clase que señalan es heredera, por ejemplo *Elemento* es subclase de *Agent*.

También por simplificar se ha omitido la clase *Funciones*, de la que todos los agentes hacen uso, que se presenta a continuación, así como la clase auxiliar Pareja.

9 Esta notación habitualmente indica estereotipos.

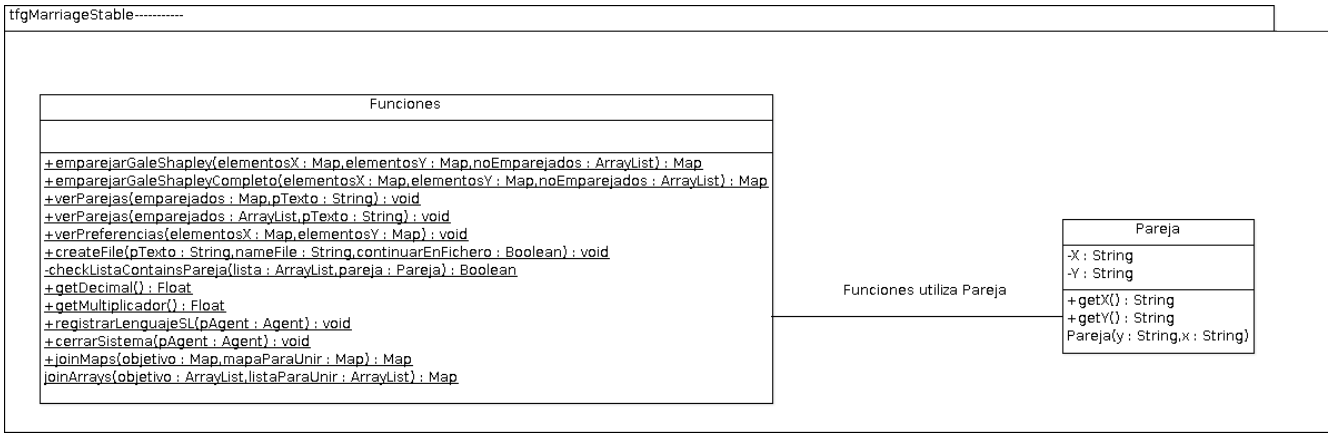

Fig. 5: Funciones

Seguidamente, se muestran los diagramas de clase de cada agente y sus comportamientos privados.

Los comportamientos relacionados con una flecha representan que desde el comportamiento origen se ha añadido el comportamiento destino de la flecha al agente. Por ejemplo, en la figura 6, desde el comportamiento *EnviarMensajesAParejas* se ha añadido el comportamiento *InitiatorMiQuery.*

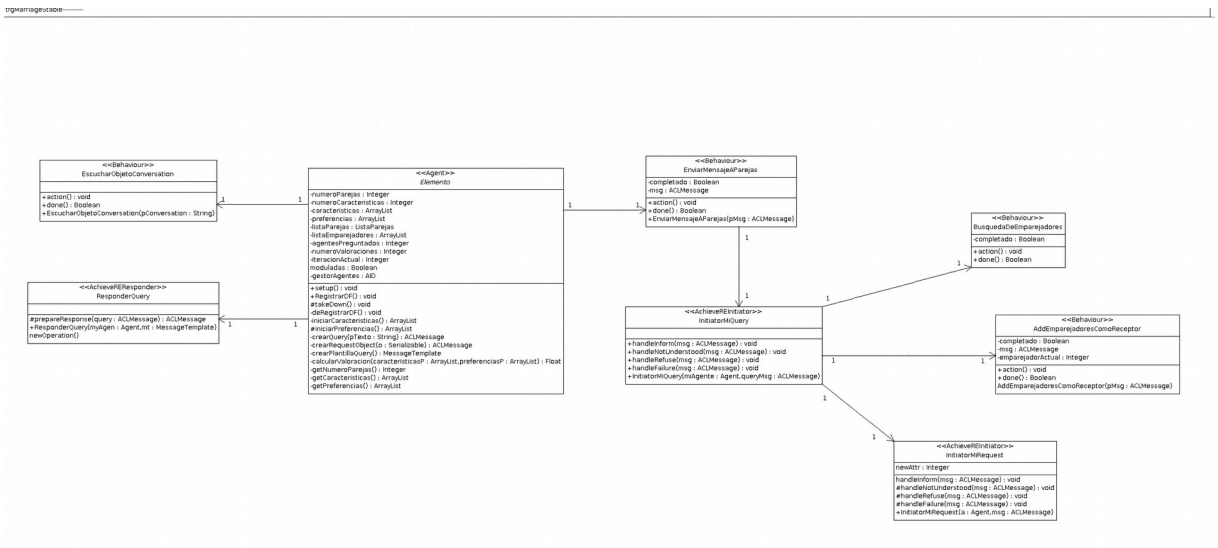

Fig. 6: Elemento

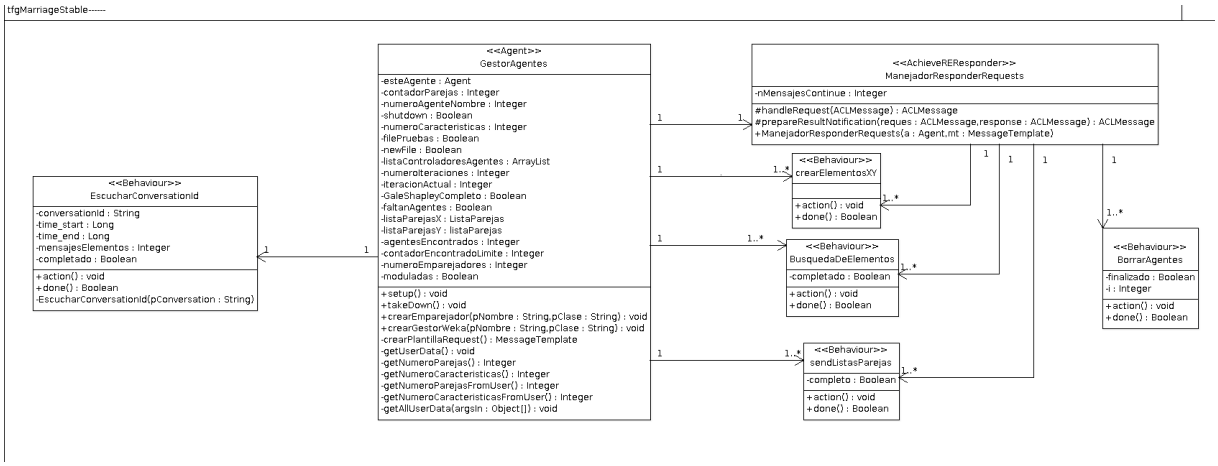

Fig. 7: GestorAgentes

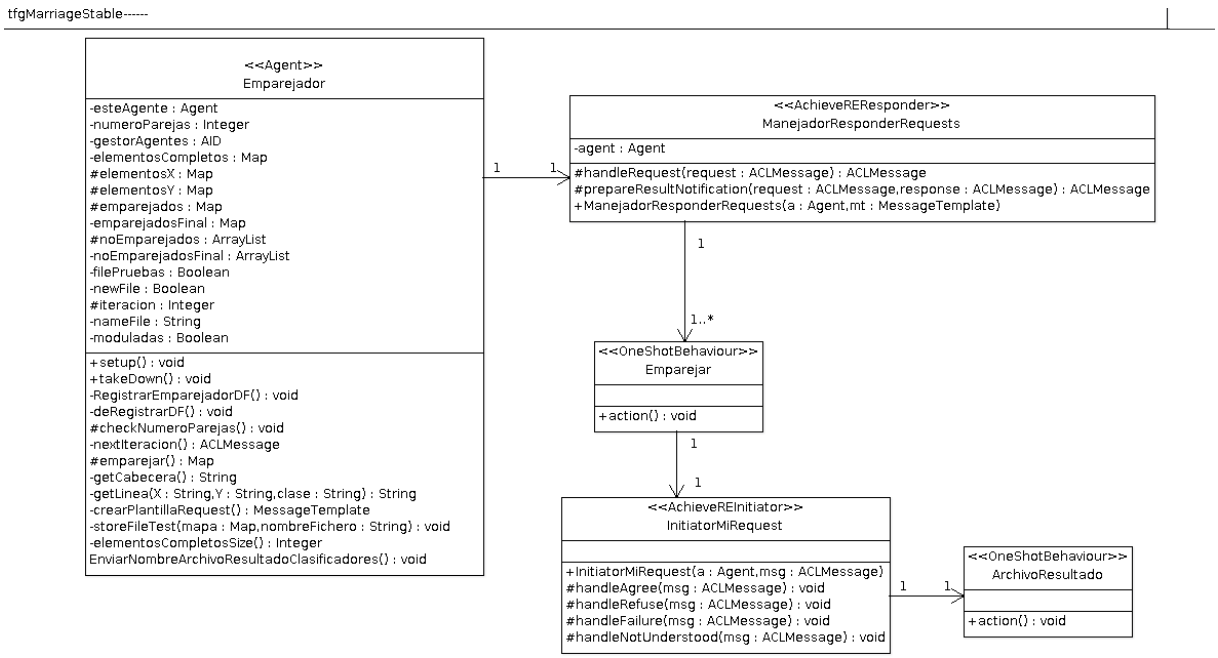

Fig. 8: Emparejador

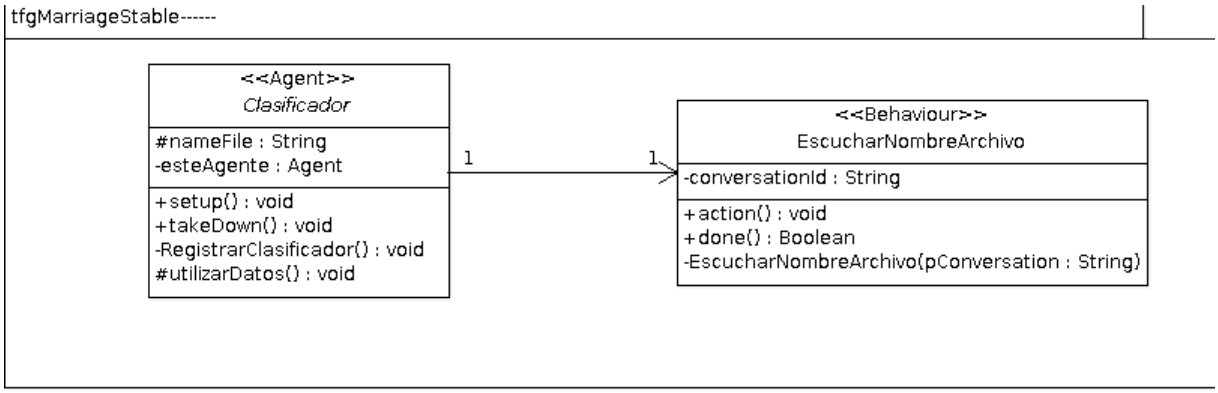

Fig. 9: Clasificador

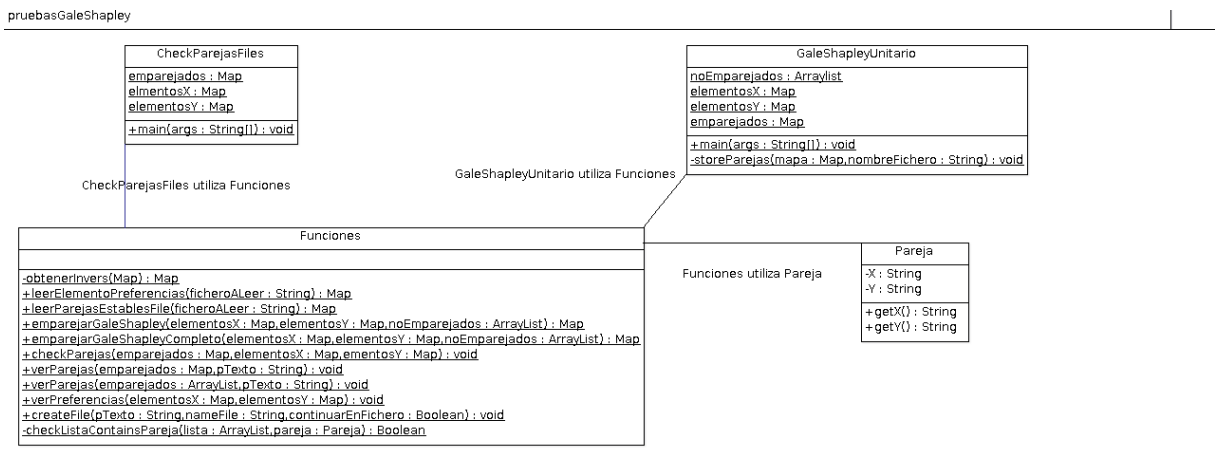

Fig. 10: Paquete pruebasGaleShapley

2.3.4 Otras consideraciones del diseño

-Elección de agentes:
Las decisiones más importantes respecto al diseño del programa han sido determinar qué agentes se ocuparían de cada tarea.

Inicialmente se tenía previsto tan solo implementar agentes de tipo *Elemento*, un *GestorAgentes* y un *Clasificador*. En aquel diseño el *GestorAgentes* se ocupaba también de realizar el emparejamiento. Sin embargo, posteriormente se consideró que el emparejado es una tarea conceptualmente diferente de la gestión de agentes y para seguir el principio de modularidad de la programación orientada a objetos resulta más adecuado separarla. Además, este diseño facilita la utilización, si se desea, de diferentes algoritmos de emparejamiento, ya que solo habría que crear las clases hijas que los implementen, como se ha hecho utilizando dos versiones del algoritmo de Gale-Shapley.

También para facilitar la reutilización del código se ha separado la función de clasificación en dos clases, *Clasificador* y *Weka*, su clase hija, donde el primero tan solo recibe el nombre del archivo de datos y es el segundo el que los procesa y analiza. Esto permite que, si se desea, se puedan incorporar fácilmente otros algoritmos clasificadores, incluso de bibliotecas ajenas a *Weka*.

-Elección de clasificadores

En este trabajo se ha utilizado tan solo uno, *Logistic*. Esta decisión no ha sido arbitraria sino que se realizaron una serie de pruebas preliminares con diferentes algoritmos clasificadores y se pudo comprobar que con números altos de parejas, o bien resultaron muy lentos, del orden de minutos, como *Multilayer Perceptron*, lo que directamente les descalifica como competencia del algoritmo de emparejamiento; o bien sus resultados fueron claramente inferiores a *Logistic* encontrando parejas estables

Se puede ver en las matrices de confusión del análisis de un mismo archivo de 400 parejas estables y 1820 no estables. Se ha utilizado la validación cruzada de 10 iteraciones.

Logistic	Naive Bayes	Decision Stump
=== Confusion Matrix ===	=== Confusion Matrix ===	=== Confusion Matrix ===
a b <-- classified as 111 289 | a = Estable 80 1740 | b =NoEstable	a b <-- classified as 9 391 | a = Estable 4 1816 | b = NoEstable	a b <-- classified as 0 400 | a = Estable 0 1820 | b = NoEstable
J48	**LazyKStar**	**Decision Table**
=== Confusion Matrix ===	=== Confusion Matrix ===	=== Confusion Matrix ===
a b <-- classified as 5 395 | a = Estable 32 1788 | b = NoEstable	a b <-- classified as 0 400 | a = Estable 312 1508 | b = NoEstable	a b <-- classified as 0 400 | a = Estable 0 1820 | b = NoEstable

-Problemas con números grandes de parejas

La mayor dificultad que se ha experimentado durante el desarrollo del software ha venido dada por el gran número de agentes tipo *Elemento*.

En una primera fase del desarrollo se decidió que cada *Elemento*, X o Y, sería el responsable de buscar a los elementos del otro grupo para preguntarles sus características. Sin embargo, en las pruebas iniciales se detectó que con números grandes la búsqueda simultánea por parte de muchos agentes utiliza demasiados recursos, produciéndose *timeouts* en la misma y la búsqueda se ejecutaba una y otra vez sin llegar a terminar en un tiempo razonable pudiendo llegar a bloquearse el sistema. Por ese motivo se decidió limitar el número de parejas posibles a 400, una cantidad de parejas que experimentalmente parecía funcionar correctamente y ser lo suficientemente grande como para permitir el análisis requerido.

Sin embargo, al continuar realizando pruebas con diferentes números de parejas, se comprobó que, con frecuencia, se producían errores y que, a pesar de generarse menos de 400 parejas, a veces los agentes no eran capaces de encontrar todos los agentes del otro grupo.

Por tanto, se decidió cambiar de táctica y asignar la responsabilidad de búsqueda de agentes en el facilitador de dominios al gestor de agentes. Es él quien busca los elementos que ofrecen los servicios de “conocerParejasX” y “conocerParejasY” y elabora dos listas, una para cada grupo. Cuando completa ambas listas envía la correspondiente a cada elemento de los grupos. De esta manera, se produce una sola búsqueda, no *2*Número de parejas*.

Con este enfoque se consiguió que no se produjeran errores con menos de 400 parejas, pero se ha mantenido este límite. El motivo es que el número de mensajes

que se han de intercambiar es *Número de parejas*Número de parejas*2*, es decir, crece con el cuadrado del número de parejas y llega a sobrecargar la capacidad de los agentes para gestionarlos simultáneamente.

-Uso de ontologías

Se ha decidido no implementar ninguna ontología para las comunicaciones entre agentes. Aunque su uso facilita la interpretación humana de los mensajes y favorece la reutilización ya que nuevos agentes podrían hacer uso de ella, se pudo constatar que su diseño y creación habrían retrasado ineludiblemente los plazos previstos en el plan de trabajo y se han sacrificado sus ventajas en pro del cumplimiento del calendario.

-Uso de archivos de registro de resultados

La idea inicial del proyecto contemplaba que el *Emparejador* suministrara al *Clasificador* los datos en forma de instancias mediante mensajes a medida que determinaba si una pareja era estable o no. Sin embargo, durante el desarrollo del código se desechó ese enfoque y se decidió que el *Emparejador* guardara la información de las instancias, es decir, las parejas estables y no estables, en un archivo.

En primer lugar, resulta más sencillo implementar la lectura del archivo desde el *Clasificador* que la transmisión continua de mensajes, pero además si se conserva e el archivo se puede reutilizar, por ejemplo como conjunto de prueba para diferentes conjuntos de entrenamiento.

Esta estrategia de conservar la información generada se amplió a guardar registros del tiempo empleado en los emparejamientos y en la formación de las listas de favoritos de los elementos.

2.4 Estudio de resultados

El objetivo del presente trabajo es determinar si un algoritmo clasificador, en concreto ***Logistic* de Weka**, es capaz de determinar más rápidamente que el algoritmo de Gale-Shapley un número aceptable de parejas estables y si, potencialmente, podría ser un sustituto eficaz. Para ello se hará uso de dos archivos que nuestro software genera o sobrescribe durante cada ejecución.

- tiemposEmparejamientos.csv: Registra el número de parejas que se han emparejado, los milisegundos que se ha tardado en el proceso, la iteración en que se ha producido y si se ha activado la opción “completo” o no.

- resultados.csv: Se actualiza en cada ejecución de la clasificación *Weka* y registra las características de los conjuntos de entrenamiento y prueba utilizados así como los resultados más importantes de la evaluación.

Recordemos que el agente *Weka* tiene dos modos de ejecución[10]. En el modo normal, el programa se inicia con la creación de parejas, se produce un emparejado que realiza el agente *Emparejador* cuyo resultado es guardado en un archivo CSV que es leído por el agente *Weka,* quien utiliza el clasificador *Logistic* para efectuar la clasificación y la evalúa sobre ese mismo conjunto de datos efectuando una validación cruzada. Sin embargo, el usuario puede también ejecutar el agente de forma directa y leer dos archivos generados previamente, utilizando el primero como conjunto de test y el segundo como conjunto de prueba.

En cualquiera de ambos modos el agente *Weka* guarda los datos de la clasificación en el archivo resultados.csv. En el apartado 2.4.2.1 *Archivo con resultados* se verán todos los parámetros que conserva ese archivo.

El archivo de resultados que se entrega representa 105 ejecuciones del agente *Weka(Logistic)* con diferentes conjuntos de entrenamiento y prueba[11] generados previamente en distintas ejecuciones del agente GestorAgentes. Cada una de esas ejecuciones se ha realizado con diferentes parámetros y el objetivo es determinar cuáles de esos parámetros dan un mejor resultado.

Para el análisis se utilizará el paquete de ofimática de LibreOffice, en concreto la herramienta de hojas de cálculo Calc. Para ello se han convertido los archivos CSV que genera el sistema al formato de Calc ODS. [12]

2.4.1 Análisis de tiempos

Como se ha señalado, el principal inconveniente del algoritmo de Gale-Shapley es que ha de examinar un gran número de parejas para determinar si son estables o no. Naturalmente, a medida que el número de parejas crece, también lo hace el tiempo que tardan en ser emparejadas, pero ¿cómo es ese crecimiento?

Para comprobarlo se ha utilizado el documento tiempoEmparejamientos.ods.

En primer lugar, se ha creado una nueva hoja en el documento llamada Análisis. En esa hoja se han copiado los números de parejas y los tiempos empleados en el emparejamiento pero, para evitar distorsiones producidas por la introducción de datos en los mapas que guardan los datos de diversas iteraciones y las parejas no estables, solo los que se han producido en la primera iteración y sin la versión completa .

10 Ver 2.3.1.1 Agentes principales/Clasificador(subClaseWeka) y también Anexo1:Manual de usuario6.1.3.3.1 Opciones Avanzadas (añadir argumentos)/Clasificador

11 No se adjuntan los archivos de estos conjuntos por su gran volumen. Por ejemplo, el archivo con el emparejamiento GaleShapleyCompleto de 400 parejas y 3 iteraciones ocupa más de 20 megas. El usuario puede utilizar el programa para generar sus propios archivos y guardarlos utilizando la opción “newfile”

12 Se adjuntan en la carpeta Análisis:resultados.ods y tiemposEmparejamientos.ods

Representamos el resultado en un gráfico de dispersión (el eje horizontal es el número de parejas y el vertical el tiempo empleado en el emparejamiento).

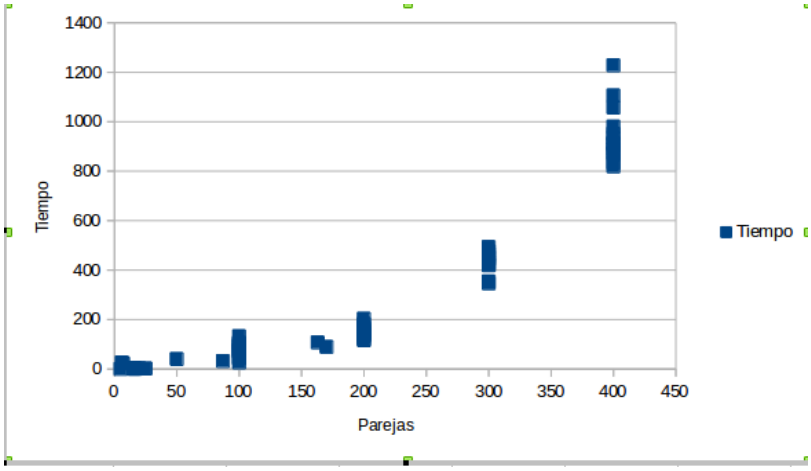

Fig. 11: Tiempos de Emparejamientos

Se puede ver que hay una gran variedad en los tiempos pero se percibe una clara tendencia de crecimiento. Calculando los tiempos medios para cada número de parejas es fácil ver que la línea de tendencia se ajusta a una curva polinómica de grado 2 y Calc suministra automáticamente su fórmula.

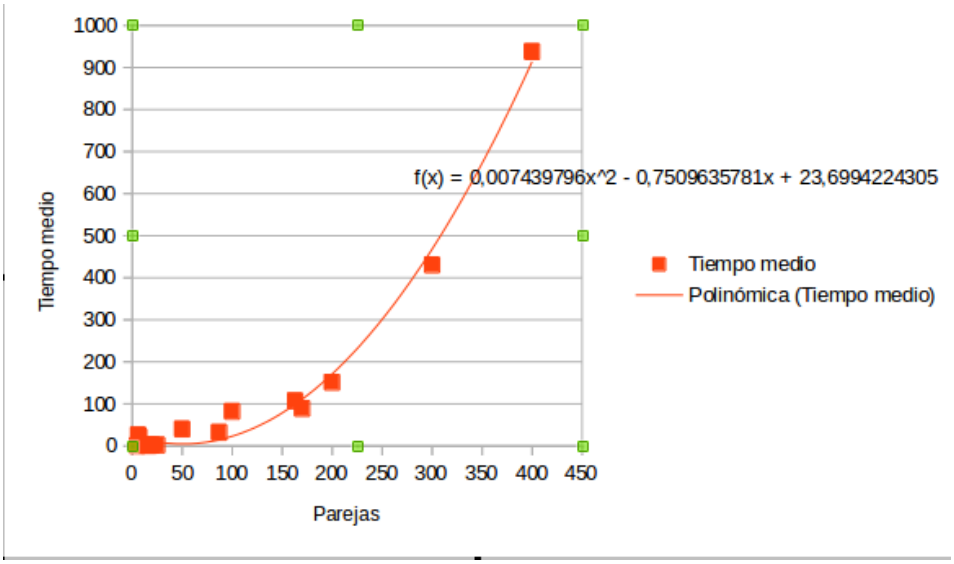

Fig. 12: Tiempos medios de emparejamientos

Se ha visto cómo se incrementan los tiempos empleados por el algoritmo de Gale-Shapley en los emparejados. A continuación se analizará cómo ha evolucionado el tiempo que ha tardado el agente *Weka* en evaluar el clasificador *Logistic* en sus diferentes ejecuciones.

En el archivo resultados.ods, dentro de la carpeta Análisis, en la hoja llamada *resultados* están todas las clasificaciones realizadas por el agente *Weka* en este estudio. Las dos primeras columnas indican qué archivos se han utilizado como

conjunto de entrenamiento y test respectivamente. Estos archivos son los que ha generado el agente *Emparejado*r en cada una de sus ejecuciones previas.

La columna *K,* “Tiempo(milisegundos) almacena el tiempo que ha tardado el agente en realizar la evaluación del clasificador *Logistic* entrenado con el archivo de la primera columna sobre el conjunto de test de la segunda o, si esta tiene el valor *null*, en efectuar la validación cruzada.

Para poder comparar los tiempos de estas evaluaciones con los tiempos de los emparejamientos ya analizados en el apartado anterior, se ha creado una nueva hoja en el documento llamada *Análisis Tiempos* para no modificar los datos originales.

En ella confrontamos el número de parejas de los diferentes conjuntos de entrenamiento frente al tiempo empleado en la evaluación del clasificador entrenado con ese mismo conjunto de entrenamiento sobre un conjunto de test (también del mismo número de parejas). Se descartan aquellas instancias que no tienen archivo de test porque en ellas se ha utilizado la validación cruzada que, inevitablemente, al realizar iteraciones lleva más tiempo.

El gráfico de dispersión es el siguiente:

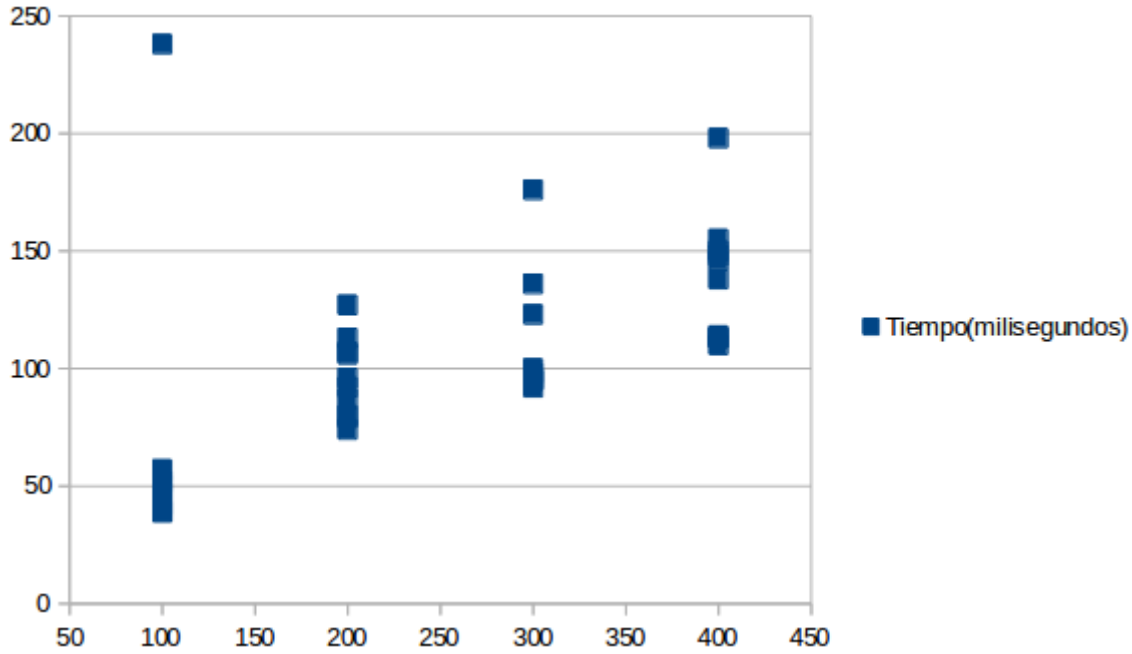

Fig. 13: Número de Parejas(Horizontal) Tiempo de evaluación (Vertical)

Aunque hay una gran dispersión,incluso uno de los tiempos de 100 parejas está completamente desplazado, se puede ver a simple vista que el tiempo empleado en realizar la evaluación de los clasificadores con números altos de parejas es menor que el empleado utilizando el algoritmo de Gale-Shapley.

Igual que anteriormente, se puede examinar qué ocurre con los tiempos promedio para cada número de parejas.

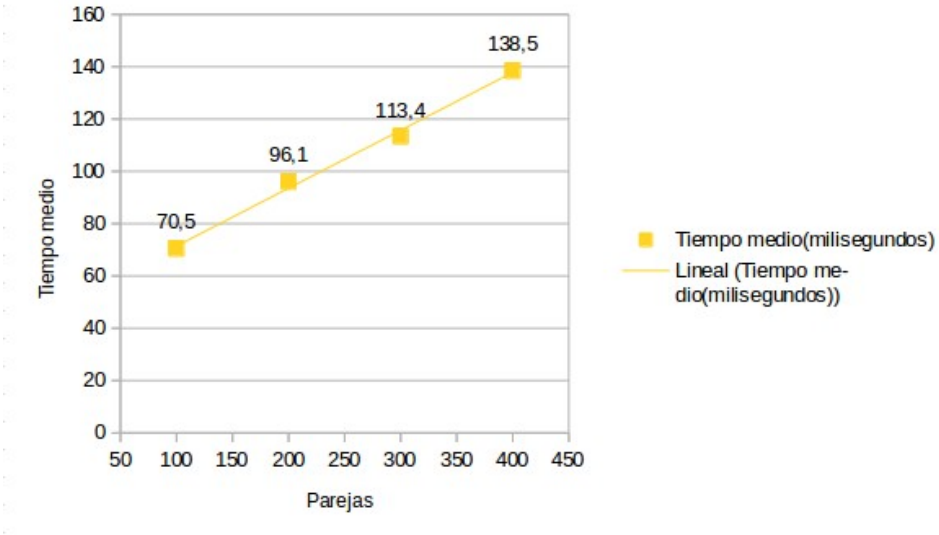

Fig. 14 Tiempos medios de evaluación de los clasificadores

En la fig.13 se ven los tiempos que se han empleado en cada evaluación frente al número de parejas. En la gráfica de la figura 14 cada punto representa el tiempo medio que han tardado todos los clasificadores de un determinado número de parejas en realizar esas evaluaciones.

En este caso el crecimiento es mucho más lento que en el caso de los emparejados y, al menos hasta 400 parejas, se aproxima a un crecimiento lineal.

Si se comparan los tiempos medios de emparejamiento(937,33 milisegundos) y de evaluación de los clasificadores (138,5 milisegundos) para un mismo número de parejas, 400, se constata que la evaluación de un clasificador es mucho más rápida que el algoritmo de emparejamiento.

2.4.2 Análisis de eficacia

2.4.2.1 Archivo con resultados

Cada vez que el agente *Weka* realiza una clasificación y su evaluación actualiza el archivo resultados.csv y guarda en cada línea los siguientes datos:

- Training(nombre Archivo): Nombre del archivo de entrenamiento
- Test(nombreArchivo):Nombre del archivo de prueba
- Parejas: Número de parejas estables del archivo de entrenamiento
- Iteraciones: Número de iteraciones realizadas al crear el archivo de entrenamiento

- Completo: Completo o NoCompleto. Indica si se ha utilizado la versión completa o no del algoritmo de Gale-Shapley al crear el archivo de entrenamiento.
- Modulación:Indica si se ha activado la opción modulación o no al crear el archivo de entrenamiento
- Parejas: Número de parejas estables del archivo de prueba, si existe
- Iteraciones: Número de iteraciones realizadas al crear el archivo de prueba, si existe
- Completo: Completo o NoCompleto. Indica si se ha utilizado la versión completa o no del algoritmo de Gale-Shapley al crear el archivo de prueba, si existe.
- Modulación:Indica si se ha activado la opción modulación o no al crear el archivo de prueba, si existe.
- Tiempo(milisegundos): Tiempo tardado en la evaluación
- Aciertos: Número de instancias correctamente clasificadas en la evaluación
- Errores: Número de instancias clasificadas erróneamente en la evaluación.
- Precisión: Tanto por ciento de aciertos totales en la evaluación
- Porcentaje de error: Tanto por ciento de fallos totales en la evaluación
- Tasa TP(Estables): Tasa de *true positive* de la categoría “Estable”. Es la tasa de aciertos en clasificar como estables sobre el total de instancias estables. Se calcula dividiendo el número de instancias correctamente clasificadas como “Estable”, entre el total de instancias que realmente son estables.
- Tasa TP(NoEstables): Tasa de *true positive* de la categoría “NoEstable”. Es la tasa de aciertos en clasificar como no estables sobre el total de instancias no estables. Se calcula dividiendo el número de instancias correctamente clasificadas como “NoEstable” entre el total de instancias que realmente son estables.
- Precisión Estables: Tasa de aciertos de la categoría “Estable” respecto al total de elementos clasificados como estables. Se calcula dividiendo el número de instancias correctamente clasificadas como “Estable”, entre el total de instancias así clasificadas.
- Precisión No Estables: Tasa análoga, pero de la categoría “NoEstable”.
- F-Measure Estables: F-Measure de la categoría “Estable”
- F-Measure No Estables: F-Measure de la categoría “NoEstable”

- Kappa: Estadístico Kappa.

Debemos explicar el significado de los últimos campos.

F-Measure (también llamado Valor-F) [17] es una medida que combina la precisión y la exhaustividad(tasa true positives) de una clase. Su valor se calcula según la fórmula

*F-Measure=2*PE/(P+E)*

donde P es la precisión de la clase y E la exhaustividad o tasa de true positives.

Su utilidad proviene de que a medida que se mejora la precisión se pierde exhaustividad e interesa que ambas sean lo más altas posibles.

Estadístico Kappa [18]: Es una medida correctora del azar en la proporción de la concordancia entre las predicciones y las clases reales. Adopta valores de -1 a 1. Valores positivos indican que hay más concordancias de las esperables por azar, valores negativos que hay menos concordancias de las esperables por azar. Un valor de 1 indicaría total concordancia y un valor de -1 total discrepancia. El valor 0 indicaría que hay las concordancias que se obtendrían por azar.

2.4.2.2 Análisis

El archivo resultados.ods contiene los datos de 105 ejecuciones reales del agente Weka con diferentes números de parejas y parámetros de ejecución. Tan solo se ha mantenido constante el número de características, cinco, para poder utilizar el resultado de una ejecución como conjunto de entrenamiento y el de otra como conjunto de prueba.

Esto permite examinar distintas clasificaciones y analizar cuáles han sido más exitosas, en función de si se ha empleado el modo completo de GaleShapley o no, el número de iteraciones, si ha habido modulación y si el número de parejas influye en la calidad de un clasificador. Si se observa alguna clase de pauta se podrá extrapolar para números de parejas más grandes.

Se podría pensar que el factor fundamental para medir el éxito de una clasificación es la precisión total, pero el objetivo no es clasificar bien la mayoría de instancias, lo que se busca es exhaustividad, encontrar el mayor número posible de parejas estables. Por lo tanto, se han de examinar los valores de la tasa de true positive (Estable)

Se ha calculado su promedio y el valor obtenido es de 0,22. Eso significa que, de media, las clasificaciones solo han encontrado un 22% de las parejas estables.

Enfrentando el número de parejas del conjunto entrenamiento[13] a la tasa TP de Estable se puede examinar si hay una relación entre el número de parejas(eje horizontal) y el resultado del clasificador.

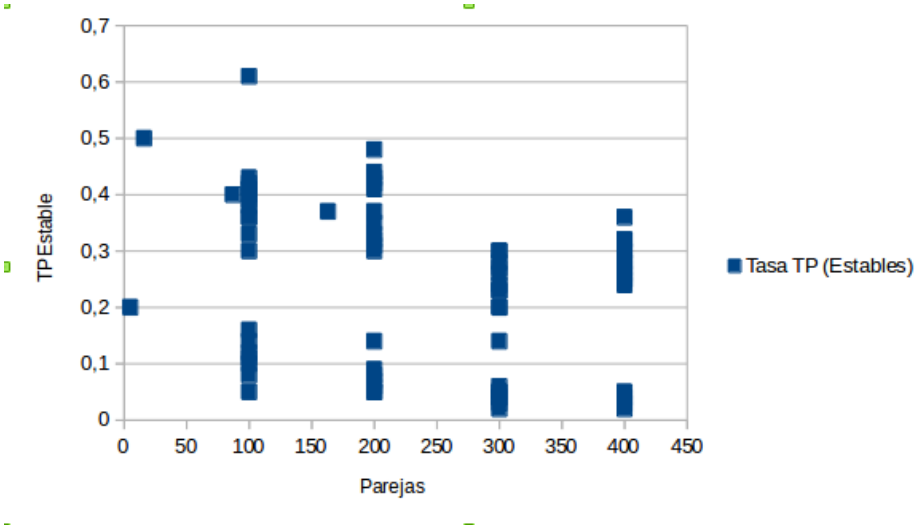

Fig. 15: TP Estable

Cada punto representa la evaluación de un clasificador con el conjunto de entrenamiento con el número de parejas indicado en el eje horizontal; en el eje vertical se marca la tasa TP(estable).

Se puede ya ver que algunos clasificadores creados a partir del mismo número de parejas lo han hecho bastante mejor que otros, de ahí las líneas verticales que forman los puntos, Para cada número de parejas hay clasificadores muy malos y otros mucho mejores. Aunque, aparentemente, los clasificadores con números de parejas más altos han bajado su eficacia, pero la gran dispersión de los datos para un mismo número de parejas no permite asegurar nada respecto a cómo evolucionaría con números de parejas mayores.

Sí se puede ya ver, en cambio, que los valores obtenidos son bastante bajos, solo en un caso se supera la tasa de 0.5, con 100 parejas, y muchos de ellos no alcanzan el 0.3.

Se puede examinar cómo varían el resto de parámetros utilizados en los clasificadores.

13 Se han filtrado aquellas instancias de las que no se tiene el número de parejas del conjunto de entrenamiento, con valor *"null"*

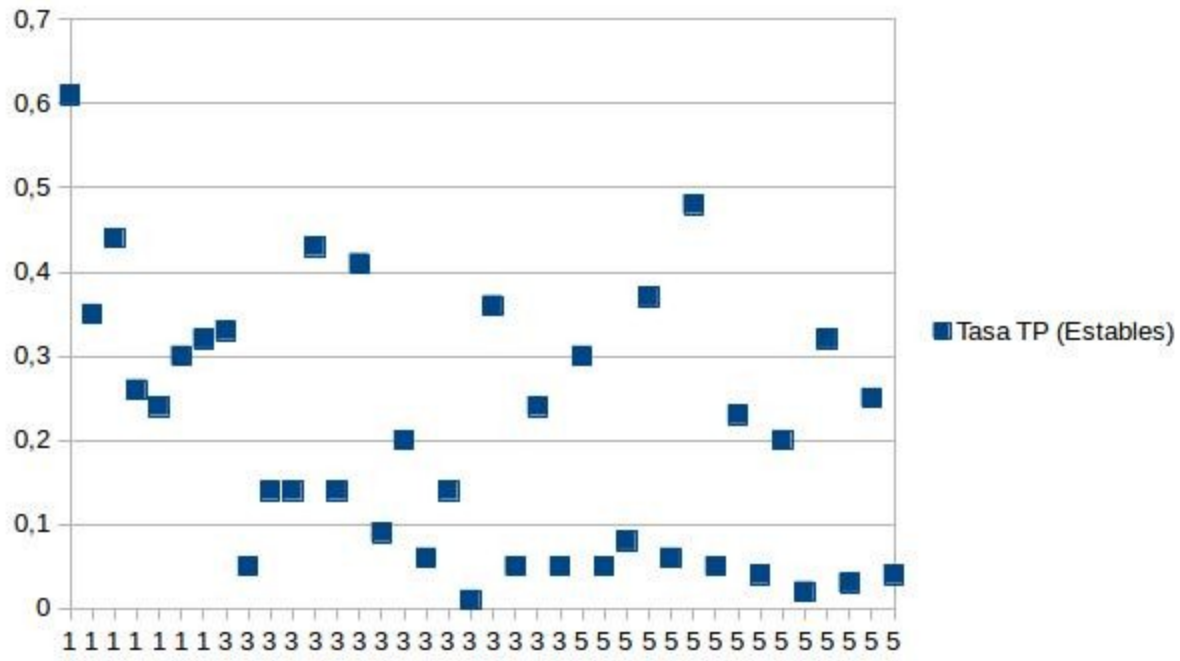

Fig. 16: Según las iteraciones

En la figura 16 cada punto representa un clasificador. En el eje horizontal se indica el número de iteraciones utilizadas en la creación del mismo, en el eje vertical su tasa TP(estables). No se aprecia una clara correlación entre ambos valores. Se puede ver que tanto cuando se han utilizado 1, 3, como 5 aparecen clasificadores mejores y peores. Por ejemplo, se podría pensar que un número alto de iteraciones es malo porque tiene varios malos clasificadores, pero es que también tiene el segundo mejor.

De forma similar, en la figura 17 se examina cómo varía la actuación de los clasificadores según su modulación. Se puede ver que tampoco hay una relación inmediata entre haberse utilizado o no y la calidad del clasificador ya que los valores tanto para *false*, como para *true* son muy parecidos.

En cambio, cuando en el eje horizontal se representa si el clasificador ha utilizado el algoritmo de Gale-Shapley Completo o no sí que se aprecia una clara diferencia, resultando los que han empleado la versión completa mucho peores. Se puede ver en la figura 18.

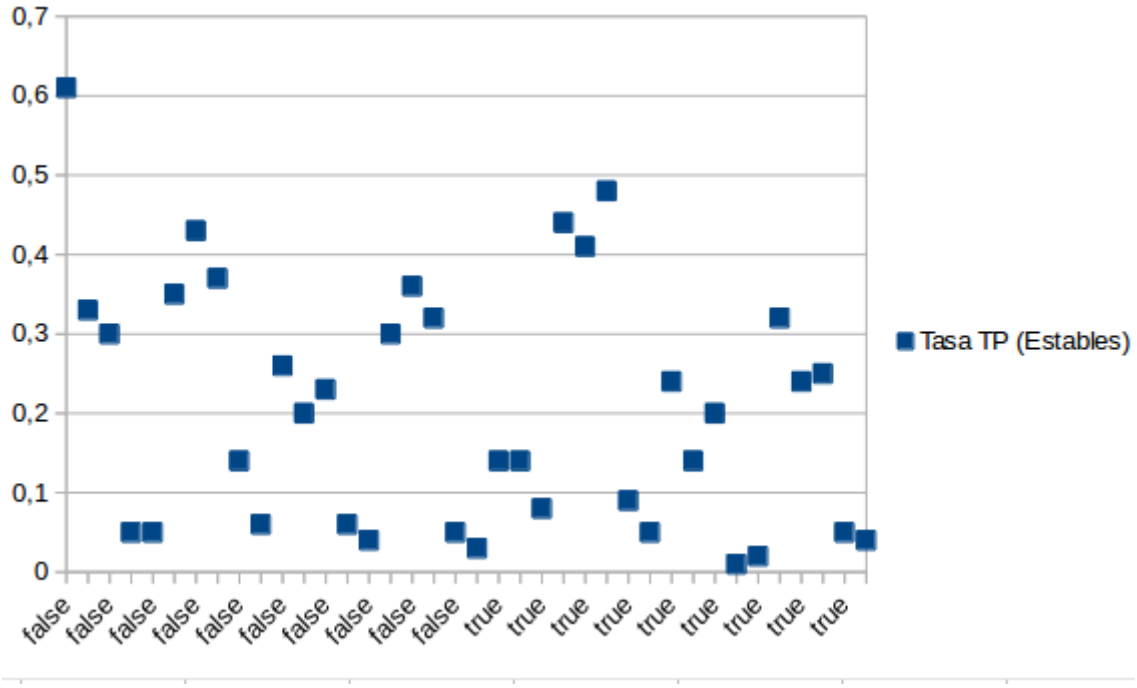

Fig. 17: Modulación

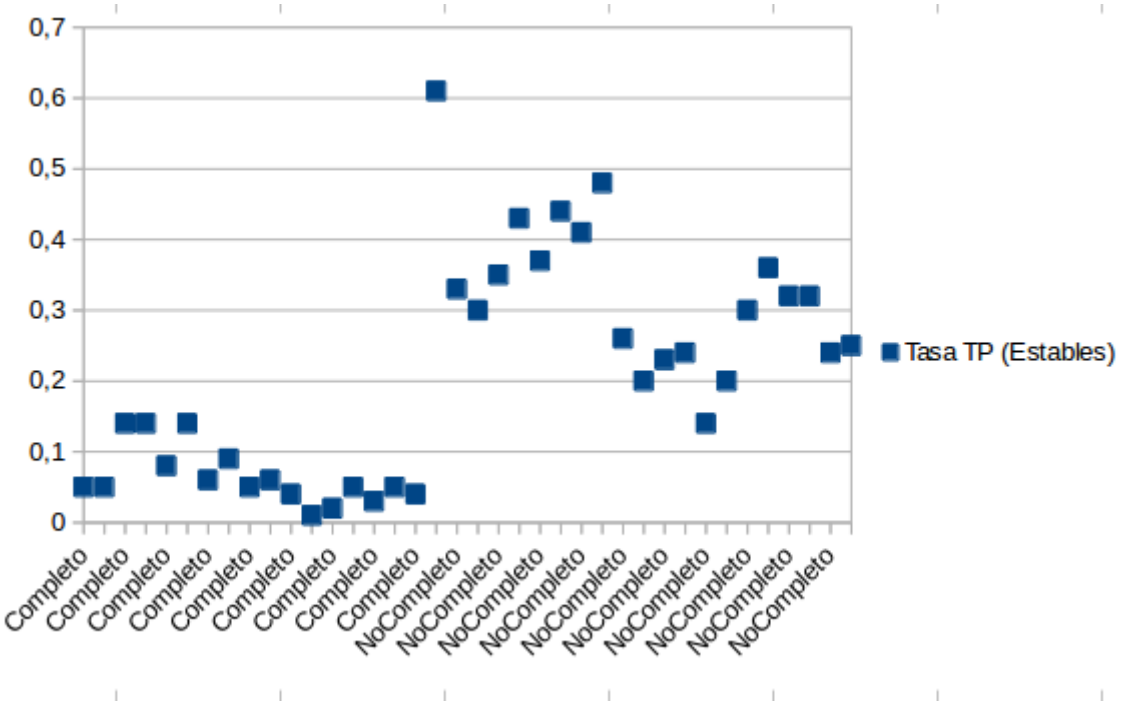

Fig. 18: Completo

Para ver más claramente qué características tienen los mejores clasificadores se examinarán los 16 mejores, según la tasa de TP (Estables), eliminando previamente aquellos en los que no se especifican los datos del conjunto de entrenamiento.

En la figura 19 se puede ver que todos ellos tienen en común que se han realizado sobre conjuntos de entrenamiento generados sin la opción de Completo. Sin embargo, ni la modulación, ni el número de iteraciones del conjunto de entrenamiento parecen determinantes. Sí en cambio se observa que que los números de parejas son bajos y que no aparecen ni las 300, ni las 400 parejas.

También se observa que la gran mayoría no han tenido un conjunto de prueba, son las filas con valores *null.* Esto significa que se ha realizado la validación cruzada y podría indicar sobreajuste, es decir, se predicen bastante bien los datos del propio conjunto de entrenamiento, pero al aplicar el clasificador a otro conjunto de datos puede que no sea así.

Parejas	Iteraciones	Completo	Modulacion	Parejas	Iteraciones	Completo	Modulacion	Tasa TP (Estables)
100	1	NoCompleto	false	100	1	Completo	false	0,61
16	1	NoCompleto	false	null	null	null	null	0,5
200	5	NoCompleto	true	200	1	Completo	true	0,48
200	1	NoCompleto	true	200	1	Completo	true	0,44
200	3	NoCompleto	false	200	1	Completo	false	0,43
100	1	NoCompleto	false	null	null	null	null	0,43
200	1	NoCompleto	true	null	null	null	null	0,42
100	1	NoCompleto	false	null	null	null	null	0,42
200	3	NoCompleto	true	200	1	Completo	true	0,41
100	1	NoCompleto	true	null	null	null	null	0,41
100	1	NoCompleto	false	null	null	null	null	0,41
87	1	NoCompleto	false	null	null	null	null	0,4
100	5	NoCompleto	false	null	null	null	null	0,4
100	3	NoCompleto	false	null	null	null	null	0,39
100	1	NoCompleto	false	null	null	null	null	0,38
200	5	NoCompleto	false	200	1	Completo	false	0,37

Fig. 19: Los 16 mejores clasificadores según la tasa TP (ESTABLES)

Ya que el objetivo es sustituir el algoritmo de Gale-Shapley con números de parejas grandes se examinarán con más detalle los resultados de los grupos de parejas más grandes, 200, 300 y 400.

Se han seleccionado asimismo aquellos en los que se sometía cada conjunto de entrenamiento de un determinado número de parejas al mismo conjunto de test (para ese número de parejas en concreto). También se han seleccionado los mismos números de iteraciones (1,3,5) y se han probado ambos tipos de modulación. De esta forma se puede estudiar mejor el impacto de cada parámetro en la calidad del clasificador.

Por último, se han escogido únicamente aquellos en los que el algoritmo empleado ha sido NoCompleto que, como se ha visto, es más eficaz.

El resultado de esta selección se puede examinar en la figura 20.

Lo primero que llama la atención es que la tasa TP tiene un rango muy bajo, entre 0.48 y 0.14.

El clasificador más exitoso, se ha producido con 200 parejas, tiene un número alto de iteraciones, y modulación *true.* Sin embargo, esto no es así cuando se examinan 400 parejas, ni tampoco con 300 parejas.

Para 400 parejas el clasificador más exitoso se produce con 3 iteraciones y modulación *false,* mientras que para 300 se obtiene con una sola iteración y modulación también *false*.

Tampoco hay una clara correlación entre el número de parejas y la calidad del clasificador porque las parejas intermedias (300) son precisamente las que han producido los peores clasificadores. Aunque, como se ha visto en la figura 19, es muy probable que a medida que el número de parejas crezca los clasificadores se vayan comportando peor.

Parejas	Iteraciones	Completo	Modulacion	Tasa TP (Estables)
200	5	NoCompleto	true	0.48
200	1	NoCompleto	true	0.44
200	3	NoCompleto	false	0.43
200	3	NoCompleto	true	0.41
200	5	NoCompleto	false	0.37
400	3	NoCompleto	false	0.36
200	1	NoCompleto	false	0.35
400	5	NoCompleto	false	0.32
400	1	NoCompleto	true	0.32
400	1	NoCompleto	false	0.30
300	1	NoCompleto	false	0.26
400	5	NoCompleto	true	0.25
400	3	NoCompleto	true	0.24
300	1	NoCompleto	true	0.24
300	5	NoCompleto	false	0.23
300	3	NoCompleto	false	0.20
300	5	NoCompleto	true	0.20
300	3	NoCompleto	true	0.14

Fig. 20 Los mejores clasificadores con números grandes de parejas

Se concluye, por tanto, que los clasificadores en modo NoCompleto son mejores, que muy probablemente los clasificadores de números de parejas altos tendrán peores resultados, pero no se puede aventurar nada sobre el resto de características que ha de tener un buen clasificador para tener tasas altas de *True Positive* (Estables).

La tasa de verdaderos positivos no es el único parámetro de interés a considerar, también es importante saber la precisión de esos aciertos.

Ampliando el cuadro de la figura 20 se puede ver que no solo las tasas de t*rue positive* son bajas sino que la precisión de estos “mejores clasificadores” son tremendamente bajas y se mueven en un rango de 0.2 a 0.31

Basta como ejemplo el clasificador con la tasa más alta. Aunque ha encontrado el 48% de las parejas estables lo ha hecho con una precisión muy baja, 0.2.

Parejas	Iteraciones	Completo	Modulacion	Parejas	Iteraciones	Completo	Modulacion	Tasa TP (Estables)	Precision Estables
200	5	NoCompleto	true	200	1	Completo	true	0,48	0,2
200	1	NoCompleto	true	200	1	Completo	true	0,44	0,2
200	3	NoCompleto	false	200	1	Completo	false	0,43	0,17
200	3	NoCompleto	true	200	1	Completo	true	0,41	0,21
200	5	NoCompleto	false	200	1	Completo	false	0,37	0,23
400	3	NoCompleto	false	400	1	Completo	false	0,36	0,17
200	1	NoCompleto	false	200	1	Completo	false	0,35	0,2
400	5	NoCompleto	false	400	1	Completo	false	0,32	0,18
400	1	NoCompleto	true	400	1	Completo	true	0,32	0,16
400	1	NoCompleto	false	400	1	Completo	false	0,3	0,16
300	1	NoCompleto	false	300	1	Completo	false	0,26	0,18
400	5	NoCompleto	true	400	1	Completo	true	0,25	0,15
300	1	NoCompleto	true	300	1	Completo	true	0,24	0,21
400	3	NoCompleto	true	400	1	Completo	true	0,24	0,18
300	5	NoCompleto	false	300	1	Completo	false	0,23	0,21
300	3	NoCompleto	false	300	1	Completo	false	0,2	0,24
300	5	NoCompleto	true	300	1	Completo	true	0,2	0,22
200	3	Completo	false	200	1	Completo	false	0,14	0,31

Fig. 21: Mejores TP y Precisión

Si se incluyen las 105 clasificaciones el valor promedio medio de la precisión de la clasificación de Estable mejora un poco, alcanzando el 44%, pero sigue siendo un mal dato, sobre todo teniendo en cuenta que la mayoría de esas clasificaciones se han realizado sin conjunto de prueba y se han validado sobre el mismo conjunto de entrenamiento con validación cruzada, lo que parece indicar sobreajuste.

No es pues sorprendente que el valor combinado de la precisión y exhaustividad de la clasificación estable, F-Measure tengan asimismo un valor promedio excesivamente bajo, de tan solo 0,26; al igual que los estadísticos Kappa que, también en promedio, tienen un valor de 0,19, solo un poco mejor que las concordancias que se producirían por azar.

Valores Medios

TP Estable	Precisión Estable	F-Measure Estable	Kappa
0,22	0,44	0,26	0,19

En resumen, los clasificadores encuentran un porcentaje de las parejas estables escaso; cuando clasifican una pareja como estable tienen además una alta

probabilidad de fallar. Para terminar, no se han podido establecer criterios claros que determinen qué características han de tener los mejores clasificadores y por tanto no se puede anticipar cómo construir un buen clasificador para números de parejas mayores.

3 Conclusiones

3.1 Conclusiones del trabajo

El objetivo principal del trabajo consiste en determinar si un algoritmo de clasificación es capaz de sustituir al algoritmo de emparejamiento estable de Gale-Shapley y dados dos elementos, junto con sus características, podrá predecir si formarán una pareja estable o no, sin necesidad de examinar el resto de elementos.

Para ello se ha desarrollado un software basado en el sistema multi agente JADE que genera, en forma de agentes, elementos de dos grupos disjuntos. Cada elemento nace con una serie de características y unas preferencias determinadas. Cada agente de un grupo se comunica con todos los agentes del otro grupo y obtiene sus características. A partir de ellas, combinándolas con sus propias preferencias, forma una lista ordenada de los elementos del otro grupo, una lista de favoritos con los que preferiría emparejarse de mayor a menor. Todos los elementos envían estas listas a otro agente, un emparejador que utiliza el algoritmo de Gale-Shapley para formar parejas estables y encontrar también parejas inestables. Guarda en archivos CSV las características y preferencias de los dos elementos de cada pareja y si esta es estable o no. Un tercer tipo de agente utiliza un algoritmo de clasificación de las bibliotecas Weka y es entrenado con esos archivos para predecir si otras parejas serán estables o no, guardando a su vez el resultado de esta clasificación en otro archivo CSV.

Utilizando las diferentes posibilidades de configuración del programa (número de parejas, GaleShapley completo o no, número de iteraciones y modulación) se han creado y analizado diferentes conjuntos de entrenamiento para tratar de determinar en qué condiciones se obtienen los mejores resultados.

Se ha visto en el apartado anterior, 2.4 Estudio de resultados, que la evaluación de los clasificadores, una vez entrenados, es mucho más rápida que los emparejamientos realizados mediante el algoritmo de Gale-Shapley. El tiempo que emplea Gale-Shapley crece de forma polinómica a medida que aumenta el número de parejas mientras que el tiempo que se emplea en la clasificación lo hace linealmente. Por tanto, si la eficacia de la clasificación fuera aceptable este ahorro de tiempo sería muy útil en entornos en los que la rapidez fuera importante.

Sin embargo, se ha visto también que no es posible realizar esa sustitución. El primer indicio se obtuvo cuando se realizó el estudio preliminar para seleccionar qué clasificador o clasificadores del catálogo de Weka se utilizarían[14]. Se probaron diferentes tipos de clasificadores obteniendo con la mayoría de ellos resultados ya muy pobres. Se escogió el clasificador *Logistic* por ser el más prometedor, pero incluso este presentaba ya entonces una tasa TP (estables) muy pequeña.

Recordemos su matriz de confusión.

```
   a    b   <-- classified as
 111  289 |    a = Estable
  80 1740 |    b =NoEstable
```

TP(estables)=111/(111+289)=0,28

Sin embargo, cabía la posibilidad de que utilizando los distintos parámetros de configuración del programa se pudiera mejorar esa tasa.

De hecho, como se ha visto, en algunos casos así ha sido obteniéndose tasas de hasta el 0.61 como se puede ver en la figura 19. Sin embargo, al estudiar los casos de parejas más grandes, figura 20, las tasas bajaron a un rango comprendido entre 0.14 y 0.48.

Tampoco, como se puede ver en la figura 21, sus precisiones son aceptables ya que ni siquiera alcanzan un 50%.

No solo eso, sino que ni siquiera se han podido establecer unas pautas para diseñar el mejor clasificador posible para números de parejas más grandes, como se ha visto en el apartado 2.4.2.2. Análisis. Ni el número de iteraciones, ni si se ha aplicado la modulación o no, han resultado factores determinantes de la calidad del clasificador. Sí, en cambio, la aplicación del algoritmo de Gale-Shapley en su modo “no completo” para obtener conjuntos de entrenamiento que ha resultado mucho más eficaz que su aplicación completa y parece muy probable que con números más altos de 400 parejas la calidad del clasificador disminuya aún más.

Todo ello conduce a la conclusión de que un sistema de clasificación no puede ser entrenado para sustituir al algoritmo de Gale-Shapley.

La causa más probable es que una pareja no es estable o inestable únicamente por las características de sus miembros, sino que depende de todo el conjunto de elementos a emparejar. Basta un pequeño cambio en cualquiera de ellos para que una pareja estable inicialmente se vuelva inestable. En estas circunstancias es extraordinariamente difícil predecir si una pareja será estable o no ya que cada conjunto es diferente.

14 Ver 2.3.4 Otras consideraciones de diseño/Elección de clasificadores

3.2 Consecución de objetivos

El objetivo principal del proyecto, responder a la pregunta si se puede sustituir el algoritmo de emparejamiento estable de Gale-Shapley con un algoritmo clasificador ha sido cumplido aunque, lamentablemente, la respuesta es que no es posible.

El resto de objetivos planteados en el punto 1.2 se trataban solo de hitos a cumplir a lo largo del desarrollo del trabajo y también se han cumplido, destacando el primer punto, conocer en profundidad el “framework” JADE, ya que es la base del software desarrollado.

Sin embargo, hemos de reconocer que no se ha explotado todo el potencial del mismo ya que el producto desarrollado funciona en una sola máquina y no aprovecha las ventajas de un funcionamiento distribuido ni se han implementado mecanismos de seguridad frente a fallos de ese sistema distribuido. La planificación no preveía una implementación distribuida por la ya de por sí alta complejidad del proyecto, pero lamentamos no haber podido abordar ese enfoque adicional.

3.3 Metodología empleada

Dados los pobres resultados obtenidos con los clasificadores entrenados en cuanto a tasas de *true positives* y precisiones, cabe cuestionar si la metodología utilizada ha sido adecuada.

Ya se ha avanzado que, en nuestra opinión, la causa de esos valores tan bajos no es otra que la inherente imposibilidad de predecir con un alto grado de precisión si una pareja será estable o no sin conocer el resto de miembros de los miembros de los grupos.

Sin embargo se pueden analizar las decisiones principales del proyecto y si otras alternativas podrían haber dado mejor resultado.

Para empezar, se podría haber prescindido del sistema multiagente y crear los elementos, emparejadores y clasificador simplemente como instancias de clases, sin necesidad de comunicarse entre sí. Este enfoque posiblemente habría sido más rápido en la generación de los elementos y la transmisión de información entre ellos pero no habrían cambiado en exceso los resultados que se habrían obtenido ya que los elementos creados habrían tenido las mismas características y se habrían emparejado de la misma forma.

Si se hubiera utilizado un entorno distribuido en el sistema multiagente, es factible que al utilizar los recursos de varias máquinas el proceso de creación y comunicación entre los diferentes elementos de las parejas hubiera sido más rápido, pero también podría haber sido más lento, dependiendo de la velocidad de las diferentes máquinas y la velocidad de transmisión entre ellas. En cualquier caso, tampoco habrían variado las parejas que se habrían formado con unas mismas características ni, tampoco los resultados de los clasificadores. No obstante, es

posible que al necesitar menos recursos por máquina, se hubieran podido utilizar números de parejas más altos y examinar rangos más amplios.

Un factor que sí podría haber causado unas precisiones y tasas TP(estables) tan bajos sería la aleatoriedad de los elementos creados. Quizá ésta haya podido ser excesiva y los elementos formados en cada ejecución son demasiado diferentes entre sí hasta el punto de que evaluar un clasificador creado con un determinado conjunto de entrenamiento sobre un conjunto de prueba generado en otra ejecución diferente sea evaluar conjuntos que no tienen nada que ver entre sí.

Para mitigar esa posibilidad se diseñó la opción de modulación que disminuye en cierto grado la aleatoriedad de los elementos formados, pero quizá no lo suficiente ya que no se ha constatado ninguna mejora sustancial al usarla.

Se podría haber optado por utilizar datos del mundo real, en lugar de generar los elementos a emparejar, pero a las dificultades de obtener esos datos se añade que no sería posible generalizar los resultados a diferentes dominios del mundo real, teniendo que examinar varios de ellos.

También el análisis de los datos se podría haber realizado de diferente forma. El presente análisis se ha basado en un número de 105 ejecuciones distintas centrándose especialmente en las que se han efectuado con números grandes de parejas (200,300,400), con los mismos números de iteraciones (1,3,5) y con modulación y sin ella. Se podrían haber analizado una mayor variedad de número de parejas y del número de iteraciones.

Sin embargo, en nuestra opinión, esos datos adicionales no habrían variado demasiado los resultados ya que si con otros parámetros de uso se pudieran obtener mejores precisiones y exhaustividad se debería haber detectado algún tipo de correlación entre la variación de los parámetros de ejecución y la calidad de la clasificación.

Por supuesto, también hay que considerar si se ha utilizado el clasificador Weka correcto. Como ya se señaló[15], el estudio preliminar de clasificadores determinó *Logistic* como el más prometedor. Pero eso no quiere decir que, necesariamente, otros clasificadores no pudieran arrojar otros resultados mejores, por improbable que sea a priori.

Quizá el mayor error cometido haya sido sobrestimar la potencia de los algoritmos clasificadores y menospreciar el hecho de que un emparejado estable depende de todos los elementos de cada grupo, no solo de las características de dos elementos en particular para emparejar.

15 2.3.4 Otras consideraciones del diseño/Elección de clasificadores

3.4 Ajuste a la planificación

La planificación inicial ha seguido, en líneas generales, la previsión inicial y las fechas de entregas parciales se han cumplido con puntualidad con todos sus objetivos cumplidos.

Sin embargo, como es natural, algunas de las tareas intermedias han experimentado ligeras variaciones. La mayor dificultad que se ha encontrado para mantener el calendario de planificación ha venido dada por la inexperiencia en el desarrollo de software con grandes requerimientos.

En primer lugar la limitación por defecto a 100 de JADE sobre el número de agentes que puede encontrar en una búsqueda ha ocasionado que se hubieran de implementar mecanismos de seguridad ante la eventualidad de que el usuario intentara superar ese número corriendo el riesgo de bloquear el sistema.

Por otro lado, la comunicación entre agentes cuando el número de parejas es alto consume tantos recursos que ha obligado a modificar en dos ocasiones todo el método de búsqueda para evitar que se produjeran fallos aleatorios.

En cambio, la integración de Weka en el sistema, y el desarrollo del algoritmo de Gale-Shapley llevaron menos tiempo del previsto por lo que se pudo considerar mejorar el código del programa introduciendo el paquete de pruebas *pruebasGaleShapley* que valida y comprueba el algoritmo de Gale-Shapley de modo independiente; se pudo implementar el uso de las diferentes opciones del programa para crear distintos tipos de conjuntos de entrenamiento; y se implementó el sistema de registro automático de todas las ejecuciones para facilitar el estudio posterior

3.5 Posibles líneas futuras de ampliación

Aunque el resultado de este trabajo descarta la posibilidad de sustitución del algoritmo de Gale-Shapley por un clasificador no lo hace con los algoritmos de emparejado de modo general.

Es posible que un clasificador obtenga mejores resultados con otros tipos de algoritmos de emparejado, por ejemplo, los que utilizan *Skylines* [4].

Otra línea de ampliación posible sería mejorar el software creado para que funcione en un verdadero entorno distribuido, ya que la presente versión está diseñada para operar en una sola máquina. Eso podría permitir examinar mayores números de parejas y aumentar el rango de estudio.

Por supuesto, también se puede experimentar con otros tipos de clasificadores diferentes de Weka.

Sin embargo, se ha de advertir que es muy probable que se llegue a resultados parecidos ya que el emparejado estable sigue dependiendo del resto de elementos de los conjuntos, no solo de las características de dos elementos cualesquiera que se quiere comprobar si formarían una pareja estable o no.

4 Glosario

Agente: Agente de software, es una parte del software que actúa para un usuario u otro programa de una manera más o menos autónoma.

AID: Identificador único de un agente en un entorno JADE

Algoritmo: Procedimiento que sigue unas reglas sistemáticas para realizar un cálculo o resolver un problema.

Array: Arreglo. Estructura ordenada de datos que agrupa elementos del mismo tipo y que permite su recuperación mediante un número de índice.

ASM (Agent Management System): Agente que registra las direcciones de todos los agentes dados de alta en un contenedor principal de JADE

CDN (Content Delivery Network): Red que contiene copias de datos para optimizar el acceso a los mismos.

Clasificador: Software que dados unos datos trata de determinar dentro de un conjunto de categorías a cuál de ellas pertenece una o varias muestras.

*Cluste*r: Del inglés, grupo, racimo. Grupo de computadoras que conectadas entre sí se comportan como una sola

CSV (*comma-separated values):* Tipo de documento legible por el ser humano que guarda datos en forma de tabla en la que las columnas se separan por comas y las filas por saltos de línea.

DF (*Domain Facilitator*): Agente facilitador de dominios, que registrará los servicios que ofrecerán los diferentes agentes en un contenedor principal de JADE

Dropbox: Sistema de almacenamiento de datos en la nube que permite la sincronización con directorios locales.

FIPA (*Foundation for Intelligent Physical Agents*): Organismo que establece los estándares de comunicación entre agentes.

Float: Número en coma flotante, precisión 32 bits. Es un tipo primitivo, básico de datos en Java.

Framework: Entorno de trabajo, conjunto de herramientas de software y prácticas para resolver un determinado tipo de trabajo.

Eclipse: Un entorno de desarrollo integrado de software libre para desarrollar aplicaciones informáticas

IDE (*Integrated Development Environment*): Entorno de desarrollo integrado. Software que facilita con diversas herramientas el desarrollo de aplicaciones informáticos

HTML (*HyperText Markup Language):* lenguaje de marcas de hipertexto. Empleado sobre todo en los documentos web

JADE (*Java Agent Development Framework*): Entorno de Desarrollo de Agentes en Java.

Java: Lenguaje de programación de propósito general orientado a objetos.

Mapa clave valor: Estructura de datos de Java que permite guardar una colección de objetos identificando cada uno de una manera única con una clave única asociada a un valor.

Matriz de confusión: Representación gráfica del resultado de una clasificación de un conjunto de instancias. En una tabla se muestran en las columnas el número de instancias clasificadas en una categoría y en las filas el número de instancias que realmente pertenecen a una categoría determinadas.

Minería de datos: Campo del conocimiento que a través de la ingeniería computacional, la estadística, el aprendizaje automático y otras técnicas trata de descubrir patrones en conjuntos de datos.

Normalización: Procedimiento que ajusta conjuntos de datos de diferentes rangos para que tengan una escala común.

PAC (*Prova d'Avaluació Contínua*): Prueba de evaluación contínua.

Programación Orientada a Objetos: Paradigma de programación en el que los objetos del mundo real se representan mediante clases que agrupan sus características y funcionalidades.

Query: Consulta, un tipo de mensaje que envía un agente preguntando por una información.

Request: Petición, un tipo de mensaje que envía un agente pidiendo que se realice una determinada acción

Serializar: Proceso de codificación de un objeto que permite su almacenamiento y recuperación posterior de forma idéntica.

SMA (Sistema Multi Agente): Sistema que combina diferentes agentes de forma distribuida para realizar una tarea.

String: Cadena de caracteres, es uno de los tipos primitivos, básicos, de datos en Java.

Weka: Software de aprendizaje computacional y minería de datos desarrollado por la Universidad de Waikato, Nueva Zelanda.

5 Bibliografía

[1] *El problema de los matrimonios estables y otros problemas de emparejamiento* /Marina Blanco Peña; Universidad de la Rioja, Servicio de Publicaciones, cop. 2015
https://biblioteca.unirioja.es/tfe_e/TFE001024.pdf
[Consulta:19/11/2017]

[2] *Algorithmic Nuggets in Content Delivery* /Bruce M .Maggs,Ramesh K. Sitaram; Newsletter ACM SIGCOMM Computer Communication Volumen 45 Número 3, Páginas 52-66/ (Julio 2015)
https://dl.acm.org/citation.cfm?id=2805800
[Consulta:01/10/2017]

[3] *College Admissions and the Stability of Marriage* /D. Gale, L.S. Shapley*;*
The American Mathematical Monthly
Vol. 69, Número. 1, páginas. 9-15 (Enero 1962)
https://www.jstor.org/stable/2312726?seq=1#page_scan_tab_contents
[Consulta:28/09/2017]

[4] *SMS: Stable Matching Algorithm using Skylines* / Rohit Anurag, Arnab Bhattacharya; SSDBM '16 Proceedings of the 28th International Conference on Scientific and Statistical Database Management Article No. 24 Budapest, Hungría 18-20 Julio 2016
https://0-dl.acm.org.cataleg.uoc.edu/citation.cfm?id=2949712
[Consulta:10/10/2017]

[5] *Sampling stable marriages: why spouse-swapping won't work/* Nayantara Bhatnagar, Sam Greenberg, Dana Randall; SODA '08 Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms
páginas 1223-1232 San Francisco, California 20-22 Enero 2008
https://0-dl.acm.org.cataleg.uoc.edu/citation.cfm?id=1347215
[Consulta:10/10/2017]

[6] *Sistema multiagente*

https://es.wikipedia.org/wiki/Sistema_multiagente
[Consulta 1/10/2017]

[7] *Guia de programació de Sistemes Multiagent en JADE 3.3/* David Isern Alarcón, David Sánchez Ruenes. [Tarragona]: URV. Departament d'Engyneria Informàtica i Matemàtiques, [2005]
http://deim.urv.cat/recerca/reports/DEIM-RT-05-001.html
[Consulta 1/10/2017]

[8] Foundation for Intelligent Physical Agents
http://www.fipa.org/
[Consulta 1/10/2017]

[9] Weka 3: Data Mining Software in Java
https://www.cs.waikato.ac.nz/ml/weka/
[Consulta 15/12/2017]

[10] Eclipse.org
https://www.eclipse.org/
[Consulta 4/10/2017]

[11] Kidney Exchange Alvin E. Roth, Tayfun Sönmez and M. Utku Ünver
The Quarterly Journal of Economics Vol. 119, No. 2 (May, 2004), pp. 457-488
http://0-www.jstor.org.cataleg.uoc.edu/stable/25098691
[Consulta:29/12/2017]

[12] *Desarrollo de Agentes de software-Introducción a la tecnología de agentes/* Carmen Fernández Chamizo, Jorge Gómez Sanz, Juan Pavón Mestras. [Madrid]: Universidad Complutense de Madrid. Dep. Sistemas Informáticos y programación página3 2003
http://www.fdi.ucm.es/profesor/jpavon/doctorado/sma.pdf
[Consulta:19/11/2017]

[13] LibreOffice Calc
https://es.libreoffice.org/descubre/calc/
[Consulta 16/12/2017]

[14] Logistic
http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/Logistic.html
[Consulta 16/12/2017]

[15] Validación cruzada
https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada
[Consulta 16/12/2017]

[16] Evaluation

http://weka.sourceforge.net/doc.stable/weka/classifiers/Evaluation.html
[Consulta 16/12/2017]

[17] Valor-F
https://es.wikipedia.org/wiki/Valor-F
[Consulta 16/12/2017]

[18] Coeficiente Kappa de Cohen
https://es.wikipedia.org/wiki/Coeficiente_kappa_de_Cohen
[Consulta 16/12/2017]

[19] Download JADE
http://jade.tilab.com/download/jade/
[Consulta:17/12/017]

[20] Download Weka
https://www.cs.waikato.ac.nz/ml/weka/downloading.html
[Consulta:17/12/017]

[21] Java Developers Kit
http://www.oracle.com/technetwork/java/javase/downloads/index.html
[Consulta:17/12/017]

[22] PATH and CLASSPATH Java Tutorials
https://docs.oracle.com/javase/tutorial/essential/environment/paths.html
[Consulta:17/12/017]

[23]Hola Mundo con paquetes en Java
http://www.chuidiang.org/java/novatos/hola_mundo_paquetes.php
[Consulta:17/12/017]

[24] Programación en JADE
https://programacionjade.wikispaces.com/Agentes
[Consulta:17/12/017]

[25] JADE Class Domain
http://jade.tilab.com/doc/api/jade/domain/df.html
[Consulta:17/12/017]

Fig.2 Arquitectura JADE
http://jade.tilab.com/doc/tutorials/JADEAdmin/jadeArchitecture.html
[Consulta 18/11/2017]

6 Anexos

6.1 Anexo1: Manual de uso

6.1.1 Introducción

El propósito principal de este software es el estudio del algoritmo de Gale-Shapley de matrimonio estable y comprobar si un algoritmo clasificador puede predecir parejas estables con un grado aceptable de fiabilidad y eficiencia.
Para ello en primer lugar generará agentes de dos tipos: ElementosX y Elementos Y. El número de estos agentes será determinado por el usuario hasta un máximo de 400 parejas.

Estos agentes se comunicarán entre sí unas características generadas aleatoriamente y establecerán unas listas de preferencias por los agentes del otro grupo.

A continuación, el sistema ejecutará el algoritmo de Gale-Shapley y obtendrá, según esas listas de preferencias, un conjunto de parejas estables y no estables.
Seguidamente, utilizará un algoritmo clasificador con esos datos, y comparará los resultados de esa clasificación con las parejas estables reales obtenidas.
Por último, guardará el resultado de la clasificación en el archivo resultados.csv.

6.1.2 Requerimientos

6.1.2.1 Java Runtime Environment

Este producto, y la plataforma JADE, necesitan tener instalada el entorno de ejecución de Java. Para saber si está instalado basta con teclear java -version en la consola, o terminal del sistema.

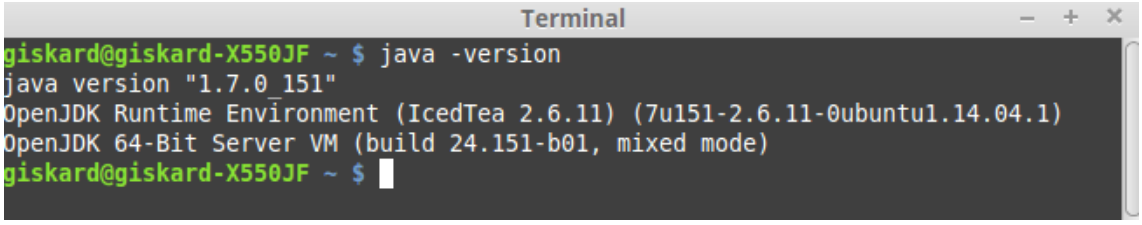

Fig. 22: Versión de Java

6.1.2.2 JADE [19]

Los paquetes de JADE se pueden descargar en la página

http://jade.tilab.com/download/jade/

Apretar "continue", aceptar la licencia "Yes, I agree" y seleccionar la versión de JADE deseada.

JADE	~ File size	Description of the content
jadeAll	17.7 MB	This file contains all JADE, i.e. it is just composed of the 4 files below. If it is too large for downloading, the 4 files below might be downloaded instead.
jadeBin	2.5 MB	This file contains JADE already compiled and ready to be used, i.e. a set of JAVA archive JAR files.
jadeDoc	12.8 MB	This file contains all the JADE documentation included the Administrator's Guide and and the Programmer's Guide. NOTICE THAT all the documentation is also available on-line.
jadeSrc	2.3 MB	This file contains all the JADE source code.
jadeExamples	490 KB	This file contains the source code of the examples and a simple demo. All the examples and demo must be compiled.

Fig. 23: Descarga JADE

Como mínimo se necesitará jadeBin, que contiene las clases compiladas.

Se trata de un archivo ZIP que contiene, comprimido en la ruta/jade/lib, un archivo: jade.jar. Se ha de extraer este archivo y guardarlo en un directorio, por ejemplo /home/user/.

6.1.2.3 Bibliotecas Weka [20]

Las bibliotecas Weka se pueden descargar en la dirección:

https://www.cs.waikato.ac.nz/ml/weka/downloading.html

Se ha de seleccionar la versión adecuada al sistema operativo utilizado y descargar el archivo.

En Linux, al igual que con JADE, se trata de un archivo ZIP que contiene comprimidas las bibliotecas.

Dentro de la primera carpeta se encuentra el archivo weka.jar. Son las bibliotecas que se necesitarán. Se han de extraer y situarlas en un directorio conocido, por ejemplo, como en el caso anterior /home/user

En cambio, en Windows se tratará de un archivo exe ejecutable que instalará todo el paquete Weka, con su interfaz gráfico incluido. Para el funcionamiento de este software se necesitará localizar el archivo weka.jar, que se encuentra en la misma carpeta donde se ha indicado que se instale Weka. Por defecto, se hará en la ruta:

C:\Program Files\Weka-3-8

Donde 3-8 es la versión de Weka instalada.

Una vez localizado el archivo hay que copiarlo y guardarlo en una localización conocida.

6.1.2.4 Java Developers Kit [21]

Esto solo será necesario si se quiere compilar el código fuente proporcionado. Si se dispone del programa ya compilado se puede omitir su instalación.

El JDK se puede descargar en la siguiente dirección:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Seguir las instrucciones de la página para la instalación según la versión elegida.

6.1.3 Compilación y ejecución

6.1.3.1 Compilación

Si solo se dispone del código fuente de los paquetes (archivos java) habrá que compilar previamente el programa para ejecutarlo. Recordemos que para ello será necesario tener instalado el Java Developers Kit (ver 6.1.2.4)

Para compilar el programa nos situaremos, desde el terminal, en la carpeta que contiene los paquetes de los archivos java: *tfgMarriageStable* y *pruebasGaleShapley*.

Se ha de crear una carpeta en la que se guardarán las clases, por ejemplo la carpeta bin

Desde la consola o terminal, si el sistema operativo es Linux se ejecutará:

```
javac -cp /home/user/jade.jar:/home/user/weka.jar -d ./bin tfgMarriageStable/*.java pruebasGaleShapley/*.java
```

Si el sistema operativo es Windows

```
javac -cp /home/user/jade.jar;/home/user/weka.jar -d ./bin tfgMarriageStable/*.java pruebasGaleShapley/*.java
```

La opción cp modifica el classpath diciéndole al compilador dónde encontrar las bibliotecas de JADE y Weka, los archivos jade.jar y weka.jar. En el ejemplo están en la ruta /home/user.

Obsérvese que la compilación en Linux y Windows se diferencian en la forma de añadir directorios y archivos al *classpath*. En Linux se añade con dos puntos «:» mientras que en Windows se utiliza el punto y coma «;».

La opción d indica dónde se guardarán las clases, en el ejemplo se guardarán en el mismo directorio donde nos encontramos(el punto) y dentro de la carpeta bin.

Para más detalles de cómo utilizar el classpath
https://docs.oracle.com/javase/tutorial/essential/environment/paths.html [22]

Para más detalles de cómo compilar un programa en java
http://www.chuidiang.org/java/novatos/hola_mundo_paquetes.php [23]

6.1.3.2 Ejecución

Una vez compilado el programa, se dispondrá de dos carpetas con un conjunto de archivos *class: tfgMarriageStable* y *pruebasGaleShapley.*

Cada paquete se ejecuta por separado. El paquete *tfgMarriageStable* constituye el verdadero programa del TFG, mientras que *pruebasGaleShapley* es solo un paquete independiente para comprobar que el algoritmo de GaleShapley implementado funciona correctamente.

6.1.3.2.1 Paquete tfgMarriageEstable

Una vez compilado el programa, se dispondrá de una carpeta tfgMarriageStable con un conjunto de archivos *class*.

Si se trabaja en Linux ejecutar:

```
java -cp .:./bin:/home/user/jade.jar:/home/user/weka.jar  jade.Boot
-jade_domain_df_maxresult 800 -gui -agents
gestor:tfgMarriageStable.GestorAgentes
```

En Windows:

```
java -cp .;./bin;/home/user/jade.jar;/home/user/weka.jar  jade.Boot
-jade_domain_df_maxresult 800 -gui -agents
gestor:tfgMarriageStable.GestorAgentes
```

Sustituyendo /home/user por la ruta en la que se encuentren los archivos jade.jar y weka.jar

La opción -cp indica el classpath, donde se buscarán las clases. Se habrá de incluir además de la ruta donde se encuentra la carpeta tfgMarriageStable (en el ejemplo está dentro de la carpeta bin), la carpeta donde están las bibliotecas JADE (en el ejemplo /home/user.jade.jar) y las bibliotecas Weka (/home/user/weka.jar)

En Linux el símbolo : permite añadir nuevos directorios al classpath
En Windows se sustituye por ;

jade.Boot indica que arrancará JADE

La opción gui inicia el interfaz gráfico de JADE

La opción agents indica los agentes que se ejecutarán al inicio:

gestor:tfgMarriageStable.GestorAgentes

-Indica el nombre del agente que se ejecutará, en este caso gestor
-tfgMarriageStable.GestorAgentes la clase a la que pertenece el agente.

Para más detalles de ejecución de JADE [24]
https://programacionjade.wikispaces.com/Agentes

NOTAS IMPORTANTES

1)
JADE limita por defecto el número de agentes que se pueden encontrar en una búsqueda a un máximo de 100 [25].

Si se desea crear más de 100 parejas, se necesita configurar el arranque del programa de forma que permita exceder ese límite. Para ello se habrá de utilizar la siguiente opción en la ejecución del agente

-jade_domain_df_maxresult <numero-deseado>

Por ejemplo, para crear 400 parejas la ejecución debería ser:

java -cp .:/home/user/jade.jar jade.Boot -jade_domain_df_maxresult 800 -gui -agents gestor:tfgMarriageStable.GestorAgentes

Obsérvese que aunque se quieren crear 400 parejas el resultado máximo se ha de marcar como 800. El motivo es que se han de buscar 400 elementos de cada grupo, en total 800. Por tanto, siempre se deberá indicar el doble, o más, que el número más alto de parejas que se pretendan crear como máximo y teniendo en cuenta que el programa, actualmente, solo admite 400 parejas como máximo.

2)
Aunque en todos los ejemplos de ejecución se ha incluido la opción *gui* que activa el interfaz gráfico de JADE, se aconseja utilizarlo solo como herramienta de

información y no para crear, o destruir los agentes que utiliza el propio sistema ya que interferiría gravemente con su funcionamiento.

6.1.3.2.2 Paquete pruebasGaleShapley

Se necesitarán los archivos: *elementosX.out, elementosY.out* y *parejas.txt*

Los dos primeros contienen elementos de cada grupo y sus listas de favoritos.

elementoX_0:elementoY_2,elementoY_3,elementoY_1,elementoY_4,elementoY_0
elementoX_1:elementoY_2,elementoY_3,elementoY_1,elementoY_4,elementoY_0
elementoX_2:elementoY_2,elementoY_3,elementoY_1,elementoY_4,elementoY_0
elementoX_3:elementoY_2,elementoY_3,elementoY_4,elementoY_1,elementoY_0
elementoX_4:elementoY_2,elementoY_3,elementoY_1,elementoY_4,elementoY_0

elementoY_4:elementoX_1,elementoX_2,elementoX_4,elementoX_3,elementoX_0
elementoY_3:elementoX_1,elementoX_2,elementoX_3,elementoX_4,elementoX_0
elementoY_2:elementoX_1,elementoX_3,elementoX_2,elementoX_4,elementoX_0
elementoY_1:elementoX_4,elementoX_1,elementoX_2,elementoX_3,elementoX_0
elementoY_0:elementoX_1,elementoX_4,elementoX_3,elementoX_0,elementoX_2

Según estos archivos, elementoX0 prefería en primer lugar emparejarse con elementoY2, si no puede con este, con elementoY3, si tampoco puede con este le gustaría con el elementoY1, seguido de elementoY4 y solo como última opción aceptaría emparejarse con elementoY0.

El resto de elementos se leerían de manera análoga.

En cuanto al archivo parejas.txt

elementoX_3:elementoY_4
elementoX_2:elementoY_3
elementoX_1:elementoY_2
elementoX_4:elementoY_1
elementoX_0:elementoY_0

Contiene una serie de parejas: elementoX_3 está emparejado con el elementoY4, X2 con Y3, etc.…

El paquete tiene dos funcionalidades:

Por un lado, a partir de los archivos *elementosX.out, elementosY.out* puede formar las parejas estables según el algoritmo de Gale-Shapley y guardarlas en un archivo *parejas.txt*.

Por otro, dados los tres archivos puede comprobar si las parejas guardadas en *parejas.txt* son estables según las preferencias establecidas en los otros dos archivos.

Formación de parejas: El usuario se situará con la consola en la carpeta que contenga los archivos *elementosX.out, elementosY.out*

Para ejecutar el programa:
java -cp ./bin pruebasGaleShapley.GaleShapleyUnitario

Como siempre, la opción cp modifica el classpath e indica la ruta en la que está la carpeta pruebasGaleShapley con los archivos class del paquete pruebasGaleShapley. En el ejemplo, los archivos *elementosX.out, elementosY.out* se encuentran en la misma carpeta que bin y dentro de bin está la carpeta pruebasGaleShapley con los archivos class.

El programa formará las parejas estables y mostrará algunas no estables:

```
PAREJAS ESTABLES:

elementoX_3,elementoY_4
elementoX_2,elementoY_3
elementoX_1,elementoY_2
elementoX_4,elementoY_1
elementoX_0,elementoY_0


 PAREJAS: 5


PAREJAS NO ESTABLES:

elementoX_0,elementoY_2
elementoX_2,elementoY_2
elementoX_3,elementoY_2
elementoX_3,elementoY_3
elementoX_4,elementoY_2
elementoX_4,elementoY_3
elementoX_0,elementoY_3
elementoX_0,elementoY_1
elementoX_0,elementoY_4


 PAREJAS: 9
```

Además, guardará las parejas estables en el archivo *parejas.txt*.

Comprobación de parejas: Nos situaremos con la consola en la carpeta que contenga los archivos *elementosX.out, elementosY.out y parejas.txt*

Para ejecutar:
java -cp ./bin pruebasGaleShapley.CheckParejasFiles

Igual que antes, la opción cp modifica el classpath e indica la ruta en la que está la carpeta pruebasGaleShapley con los archivos class del paquete pruebasGaleShapley. En el ejemplo, los archivos *elementosX.out, elementosY.out y pruebasGaleShapley.txt* se encuentran en la misma carpeta que bin y dentro de bin está la carpeta pruebasGaleShapley con los archivos class.

El resultado, si es correcto será una serie de mensajes como:

```
elementoX_0:
    elementoY_2 no quiere emparejarse con elementoX_0, prefiere a
elementoX_1
    elementoX_0, prefiere a elementoY_2 pero este prefiere a su pareja
elementoX_1

    elementoY_3 no quiere emparejarse con elementoX_0, prefiere a
elementoX_2
    elementoX_0, prefiere a elementoY_3 pero este prefiere a su pareja
elementoX_2

    elementoY_1 no quiere emparejarse con elementoX_0, prefiere a
elementoX_4
    elementoX_0, prefiere a elementoY_1 pero este prefiere a su pareja
elementoX_4

    elementoY_4 no quiere emparejarse con elementoX_0, prefiere a
elementoX_3
    elementoX_0, prefiere a elementoY_4 pero este prefiere a su pareja
elementoX_3

    elementoY_0 ya está emparejado con elementoX_0
```

Si las parejas no son estables se mostrarán mensajes como:

```
elementoX_2:
    elementoY_2 ya está emparejado con elementoX_2


    elementoY_3 no quiere emparejarse con elementoX_2, prefiere a
elementoX_1
    elementoX_2 no quiere emparejarse con elementoY_3, prefiere a
elementoY_2
ERROR: elementoY_2 y elementoX_3 preferirían estar juntos antes que con
quienes están elementoX_2, elementoY4
```

Al finalizar se contabilizan el número total de errores, parejas no estables encontradas

Opciones: Los archivos *elementosX.out, elementosY.out y pruebasGaleShapley.txt* son los archivos que los dos programas buscarán por defecto, pero se pueden sustituir por otros archivos introduciéndolos como argumentos al ejecutar los programas. Simplemente habrá que añadir las rutas y nombres de los archivos que se desee utilizar a la ejecución como argumentos. El primer archivo deberá guardará las preferencias de los elementos X, el segundo las de los elementos Y, y el tercero (solo en el caso de la comprobación) las parejas que se desee analizar.

java -cp ./bin pruebasGaleShapley.CheckParejasFiles elementosX2.out elementosY2.out parejas2.txt

Observación: El paquete pruebasGaleShapley es totalmente independiente del resto de paquetes y se puede utilizar de forma única. Tampoco hace uso de las bibliotecas de JADE y Weka.

6.1.3.3 Uso paquete tfgMarriageStable

6.1.3.3.1 Introducción de datos

Una vez que se ejecuta el programa se abre un panel de diálogo que nos preguntará el número de parejas que se quieren crear

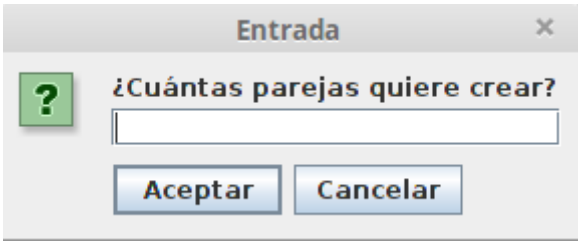

Fig. 24: Número de parejas

Se pueden introducir tantas parejas como se quiera hasta **un máximo de 400**, pero si se intenta superar ese número el sistema nos lo impedirá.

Por otro lado, si intentamos introducir un número mayor de 50 el sistema nos hará una advertencia

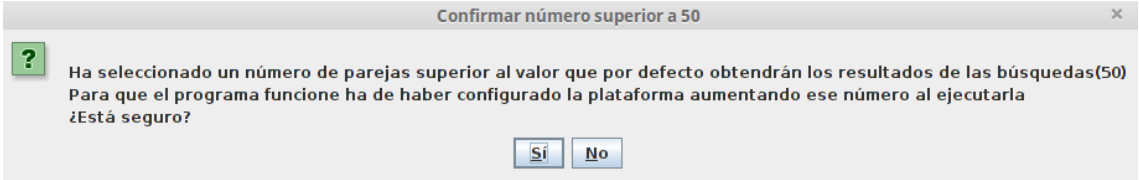

Fig. 25: Número de parejas superior a 50

Como ya se señaló[16], JADE por defecto solo encuentra hasta 100 agentes en sus búsquedas. Para superar ese número se ha de configurar al inicio de la ejecución añadiendo la siguiente opción que nos permitirá utilizar hasta 400 parejas.

-jade_domain_df_maxresult 800

Si no se incluye esta opción en la ejecución el sistema no encontrará los agentes y eventualmente causará un error y se cerrará.

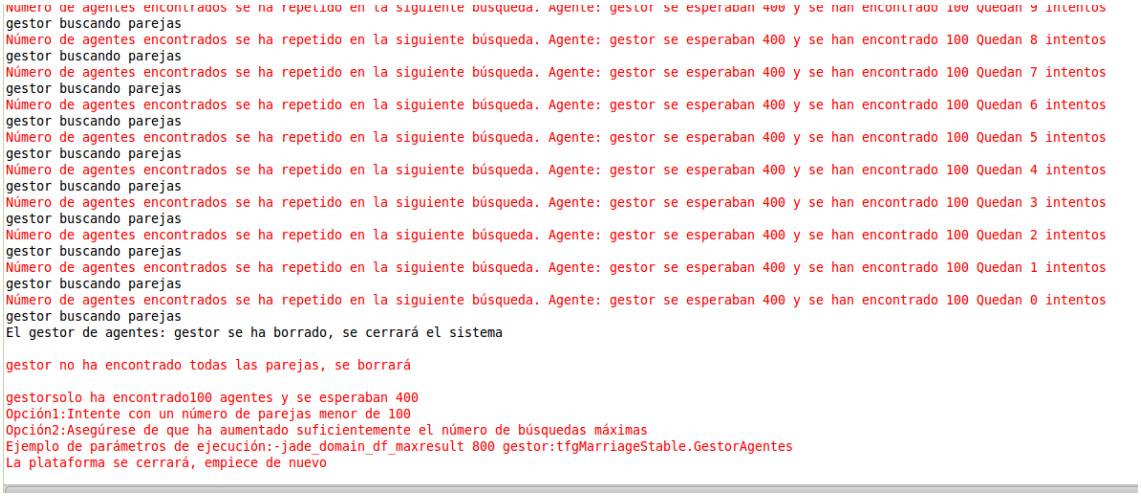

Fig. 26: Error en la búsqueda de agentes

Una vez introducidas las parejas se nos pedirá mediante otro panel introducir un número de características, entre 1 y 10

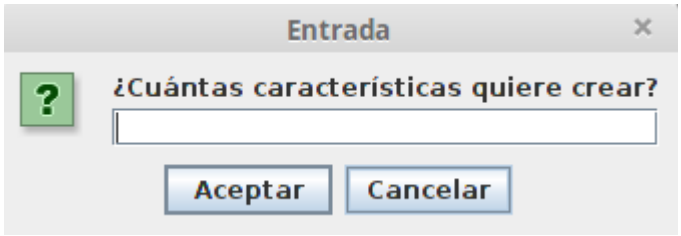

Fig. 27: Características

A continuación el sistema irá creando los diferentes agentes y mostrando cuáles son sus características y preferencias

16 6.1.3.2. Notas Importantes 1)

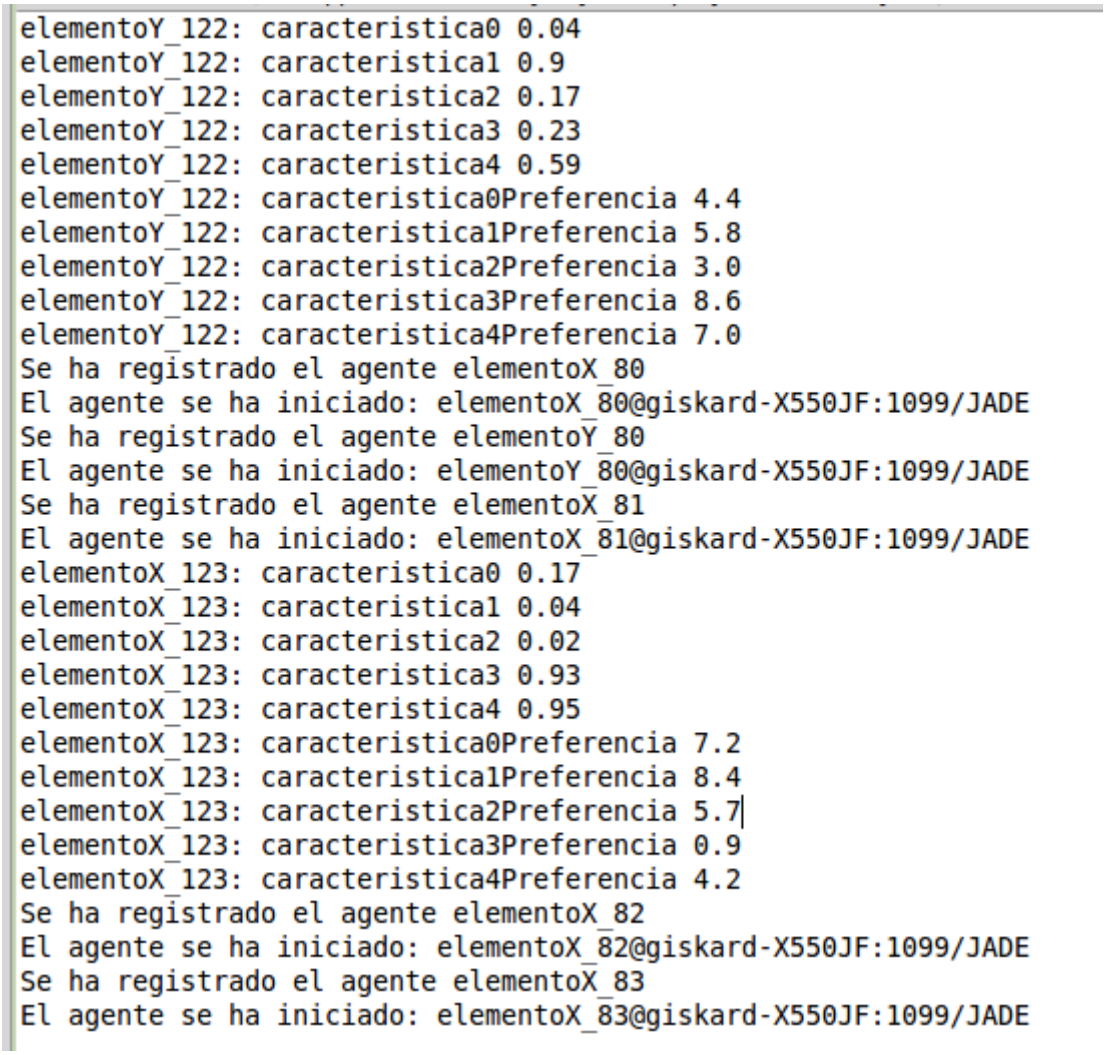

Fig. 28: Creación de agentes

En esta fase, si el número de parejas es muy alto la creación de los agentes irá cada vez más lenta al necesitar cada vez más recursos.

Una vez creados todos los elementos, el gestor de agentes realizará una búsqueda de todos ellos, les enviará un mensaje con la lista de los agentes encontrados y cada agente comenzará a comunicarse con los elementos del otro grupo preguntándoles sus características

```
elementoY_56 ha recibido petición de características del agente elementoX_88
elementoX_21 ha recibido petición de características del agente elementoY_204
elementoX_21 ha recibido petición de características del agente elementoY_128
elementoY_46 ha recibido petición de características del agente elementoX_399
elementoX_21 ha recibido petición de características del agente elementoY_108
elementoY_56 ha recibido petición de características del agente elementoX_201
elementoX_21 ha recibido petición de características del agente elementoY_12
elementoY_46 ha recibido petición de características del agente elementoX_215
elementoX_21 ha recibido petición de características del agente elementoY_185
elementoY_56 ha recibido petición de características del agente elementoX_187
elementoX_21 ha recibido petición de características del agente elementoY_304
elementoX_21 ha recibido petición de características del agente elementoY_345
elementoX_21 ha recibido petición de características del agente elementoY_396
elementoX_21 ha recibido petición de características del agente elementoY_282
elementoY_46 ha recibido petición de características del agente elementoX_352
```

Fig. 29: Petición de características

Esta fase, si se han creado muchos agentes, puede llevar algún tiempo.

Una vez que cada elemento ha recibido respuesta de todos los agentes del otro grupo, elabora su lista de favoritos según esas respuestas y la envía al agente emparejador. Cuando este ha recibido las listas de todos los elementos las muestra, comienza el emparejamiento y muestra su resultado

```
   LISTAS DE PREFERENCIAS DE LAS PAREJAS
.......................................

elementoX_0: [elementoY_1, elementoY_2, elementoY_0, elementoY_4, elementoY_3]
elementoX_1: [elementoY_1, elementoY_2, elementoY_0, elementoY_4, elementoY_3]
elementoX_2: [elementoY_2, elementoY_1, elementoY_0, elementoY_4, elementoY_3]
elementoX_3: [elementoY_2, elementoY_0, elementoY_1, elementoY_4, elementoY_3]
Agree recibido de GaleShapley para elementoY_4
elementoX_4: [elementoY_1, elementoY_2, elementoY_4, elementoY_0, elementoY_3]

.....................

elementoY_4: [elementoX_4, elementoX_2, elementoX_1, elementoX_0, elementoX_3]
elementoY_3: [elementoX_1, elementoX_4, elementoX_2, elementoX_3, elementoX_0]
elementoY_2: [elementoX_4, elementoX_2, elementoX_1, elementoX_0, elementoX_3]
elementoY_1: [elementoX_2, elementoX_4, elementoX_1, elementoX_3, elementoX_0]
elementoY_0: [elementoX_2, elementoX_4, elementoX_1, elementoX_3, elementoX_0]



Emparejando
emparejando
emparejando
emparejando
emparejando
emparejando
emparejando
emparejando
emparejando
```

Fig. 30: Lista de parejas recibidas

```
El emparejado ha tomado 0 millisegundos
SON PAREJAS ESTABLES

elementoX_0,elementoY_4
elementoX_3,elementoY_3
elementoX_2,elementoY_2
elementoX_4,elementoY_1
elementoX_1,elementoY_0


 PAREJAS: 5



SON PAREJAS INESTABLES

elementoX_0,elementoY_1
elementoX_1,elementoY_1
elementoX_3,elementoY_0


 PAREJAS: 3
```

Fig. 31: Emparejamientos

A continuación, se escribe en un archivo (por defecto Datos.csv) las parejas formadas, tanto las estables como no estables, junto con sus características. Nuevamente, si hay muchas parejas esta fase puede llevar algún tiempo.

Una vez escrito el archivo, el clasificador Weka lo lee y muestra un resumen de los datos del mismo y el resultado de su clasificación

```
=== Confusion Matrix ===

     a     b   <-- classified as
    29   271 |     a = Estable
    30 10124 |     b = NoEstable

El evaluador ha tardado: 2771 milisegundos
Archivo de entrenamiento:GaleShapleyCompleto_parejas_100_Iteraciones_3_2
Archivo de test:null
Aciertos:10153.0
Errores:301.0
Precision:97.12%
Porcentaje error:2.88%
Tasa true positives Estables:0.10
Tasa true positives No Estables:1.00
Precision Estables:0.49
Precision No Estables:0.97
F-Measure Estables:0.16
F-Measure NoEstables:0.99
Kappa estadístico:0.15
```

Fig. 32: Clasificación

La matriz de confusión indica cómo se han clasificado los datos.

Las verticales indican cómo se han clasificado las instancias y las horizontales a qué clase pertenecen realmente.

La interpretación del ejemplo es clara:

-Se han clasificado 29 instancias como estables y realmente son estables.
-Se han clasificado 30 instancias como estables pero son inestables.
-Se han clasificado 271 instancias como no estables siendo realmente estables.
-Se han clasificado 10124 instancias correctamente como no estables.

A continuación se muestra el tiempo que ha tardado el evaluador en realizar la clasificación de las instancias, en este caso 2771 milisegundos.

Seguidamente se indica qué archivos se han utilizado de entrenamiento y de prueba. Si no hay archivo de datos se ha utilizado el archivo de entrenamiento realizando una validación cruzada de diez iteraciones.

Aciertos: es el número de instancias correctamente clasificadas.

Errores: es el número de instancias incorrectamente clasificadas.

Precisión: es el porcentaje de aciertos obtenido sobre el total de instancias.

Porcentaje de error: es el porcentaje de errores sobre el total de instancias.

Tasa true positives Estables: es la tasa de aciertos en clasificar como estables sobre el total de instancias estables. En este caso 0,10 indica que se han encontrado tan solo el 10% de las parejas estables.

Tasa true positives No Estables: es el porcentaje de aciertos en clasificar como no estables sobre el total de instancias no estables. En este caso, 1,0 indica que se han encontrado el 100% de las parejas no estables (en realidad no se han encontrado el 100% se trata de un redondeo porque la tasa encontrada es 10124÷(30+10124)=0,997

Precisión estables: es la precisión de aciertos en clasificar instancias como estables respecto al total de clasificados estables. En el ejemplo, 0,49 indica que de todas las instancias clasificadas como estables solo se ha hecho correctamente en el 49% de los casos, el resto eran no estables.

Precisión No estables: es la precisión de aciertos en clasificar instancias como no estables respecto al total de clasificados no estables. En el ejemplo, 0,97 indica que de todas las instancias clasificadas como no estables s se ha hecho correctamente en el 97% de los casos, el resto eran estables.

F-Measure Estables: F-Measure (también llamado Valor-F) es una medida que combina la precisión y la exhaustividad(tasa true positives) de una clase. Su valor se calcula según la fórmula:

F-Measure=2*PE/(P+E) donde P es la precisión de la clase y E la exhaustividad o tasa de true positives.

Su utilidad proviene de que a medida que mejoramos nuestra precisión perdemos exhaustividad y nos interesa que ambas sean lo más altas posibles, una medida de esto nos la da F-Measure y un clasificador nos estará dando un mejor resultado cuanto más alto sea ese valor.

F-Measure No estables: Ídem , para la clase no estables

Kappa estadístico: Es una medida correctora del azar en la proporción de la concordancia entre las predicciones y las clases reales. Adopta valores de -1 a 1. Valores positivos indican que hay más concordancias de las esperables por azar, valores negativos que hay menos concordancias de las esperables por azar. Un valor de 1 indicaría total concordancia y un valor de -1 total discrepancia. El valor 0 indicaría que hay las concordancias que se obtendrían por azar.

6.1.3.3.1 Opciones avanzadas (añadir argumentos)

Lo explicado hasta ahora es el funcionamiento automático del programa. Sin embargo, podemos añadir algunos parámetros y modificar cómo trabaja.

El proceso de añadir argumentos es muy sencillo. Se efectúa en la ejecución del programa y consiste simplemente en añadirlos, separados por comas y entre paréntesis. Además, la clase del agente será rodeada por comillas de esta forma[17]:

java -cp .:./bin:/home/user/jade.jar:/home/user/weka.jar jade.Boot -gui -agents

“gestor:tfgMarriageStable.GestorAgentes(argumento1,argumento2)”

GestorAgentes

Los argumentos de GestorAgentes se pueden introducir en cualquier orden y pueden añadirse todos, solo algunos de ellos o ninguno.

Para ver los argumentos admitidos al crear un **gestor de agentes** consultar la sección 2.3.1.2 Opciones de ejecución

Clasificador

También, en lugar de generar los elementos X e Y y emparejarlos, se puede estudiar directamente el resultado de esos emparejamientos con nuestro clasificador. Para ello, en lugar de ejecutar el programa creando un agente GestorAgentes lo haremos creando un agente Weka

Basta con sustituir GestorAgentes por Weka

java -cp .:./bin:/home/giskard/jade.jar:/home/giskard/weka.jar jade.Boot -gui -agents

“gestor:tfgMarriageStable.**Weka**(argumento1,argumento2)”

Admite dos argumentos. El primero sería la ruta y nombre del archivo que utilizaremos como entrenamiento (obligatorio) y opcionalmente la ruta y nombre de un segundo archivo que utilizaremos como test.

17 El color y el tamaño aumentado son únicamente para señalar los elementos nuevos. No se han de incorporar en el texto que se introduce en la consola.

6.1.3.3.2 Parada del sistema

Existen varias formas de parar el sistema:

- Si se ha ejecutado el interfaz gráfico (con la opción -gui) bastará ir a File/Shut Down Agent Platform

- Si se está ejecutando desde consola se puede teclear ctrl+C lo que parará el sistema. También, cerrando la ventana de la consola.

6.1.4 Eclipse

El programa se ha desarrollado en Java, en Eclipse 3.8.1. En esta sección se explicará cómo importarlo, ejecutarlo e introducir los parámetros

El programa Eclipse y su documentación se pueden encontrar en la web

http://www.eclipse.org/

6.1.4.1 Importar el código

En primer lugar se ha de crear un nuevo proyecto, para ello se ha de ir a *File/New/Java Project.* Se le asigna un nombre un nombre, en este caso *MatrimonioEstable y* se aprieta *Finish.*

A continuación se ha de seleccionar el recién creado proyecto, e ir a *File/Import*. En la ventana que se abre, seleccionar *FileSystem* y apretar *Next.*

En el explorador del sistema se busca la carpeta donde se tenga el programa descargado (en este caso la carpeta TFG) y se marca que incluya todos los archivos y apretar Finish. Si el sistema pregunta si hay que reescribir algún archivo le diremos que sí.

6.1.4.2 Configuración del Build Path

Se necesitan s añadir las bibliotecas JADE y Weka al código, eso significa configurar el *Build Path*.

Seleccionar el proyecto, botón derecho del ratón *Build Path/Configure Build Path.*

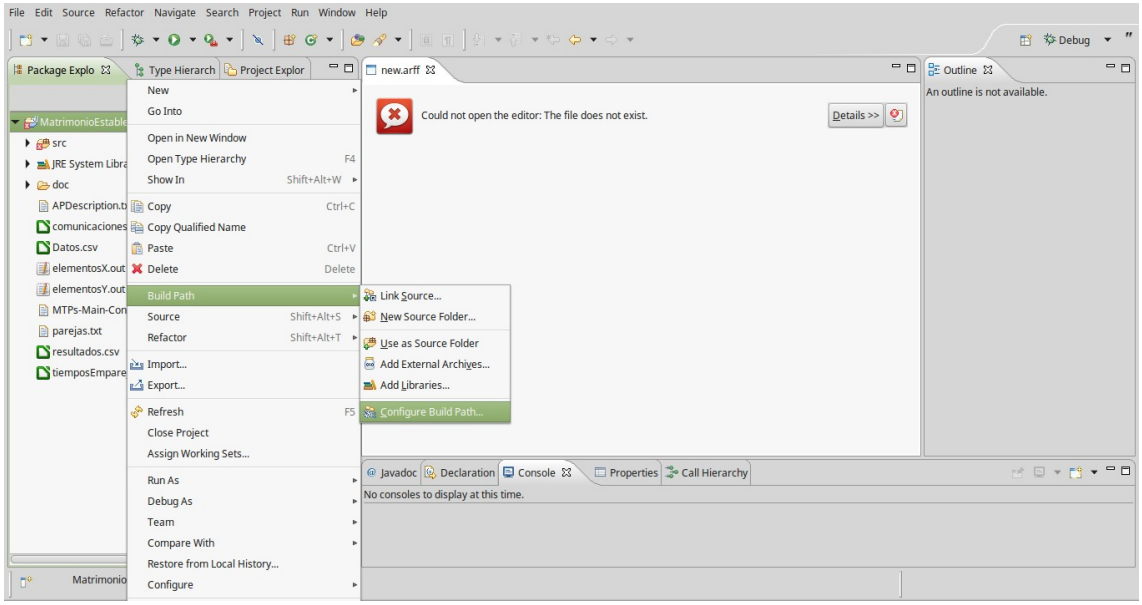

Fig. 33: Build Path

En la siguiente pantalla seleccionar la pestaña Libraries y Add External JARs

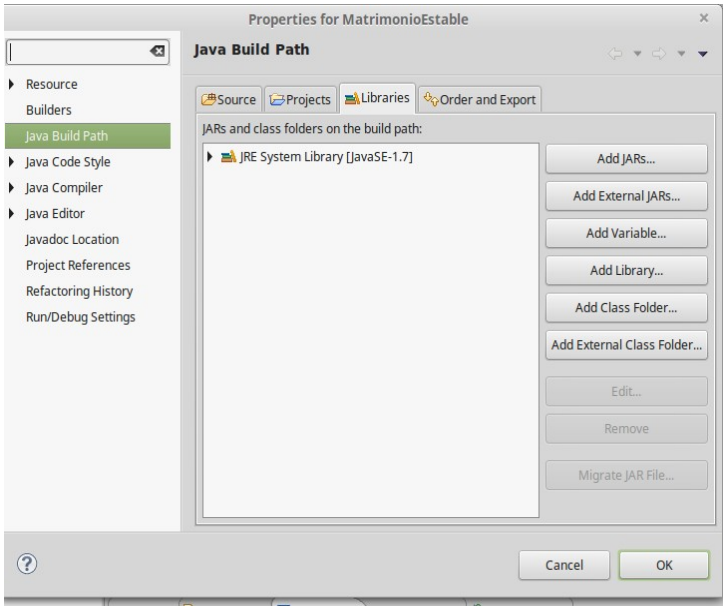

Fig. 34: Add External JARs

En el explorador del sistema que aparece buscar la carpeta en la que se han guardado las bibliotecas jade.jar y weka.jar y se añaden.

6.1.4.3 Ejecución del programa principal

En el explorador de paquetes seleccionar el paquete *tfgMarriageStable.* Apretar el botón derecho y seleccionar *Run as/Run Configurations*

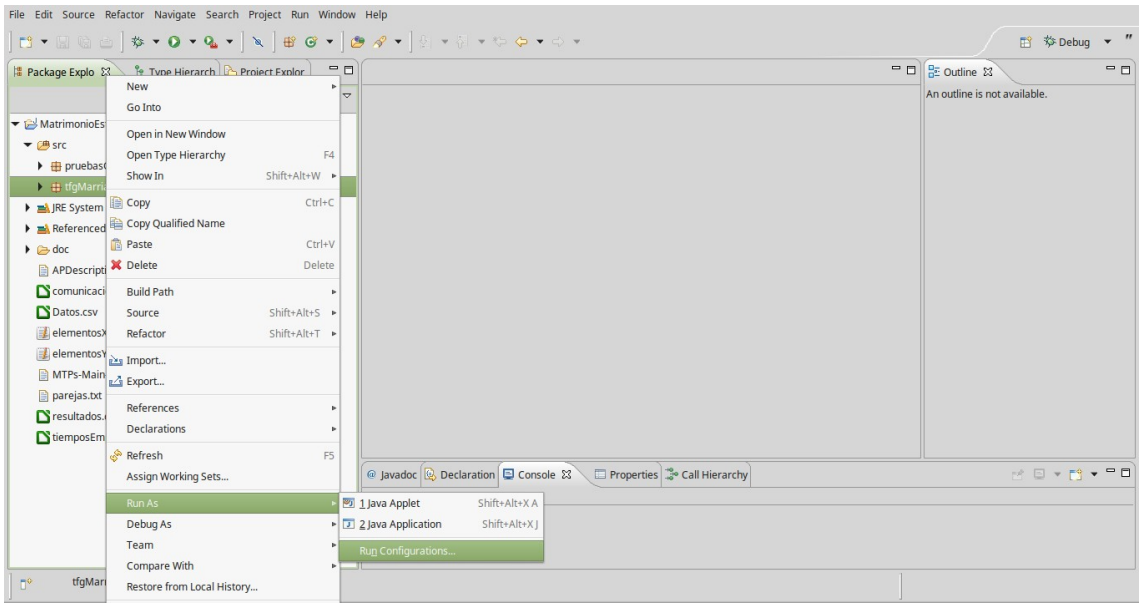

Fig. 35: Run Configurations

En la siguiente ventana seleccionar *Java Application* y apretar el botón de *new launch configuration.*

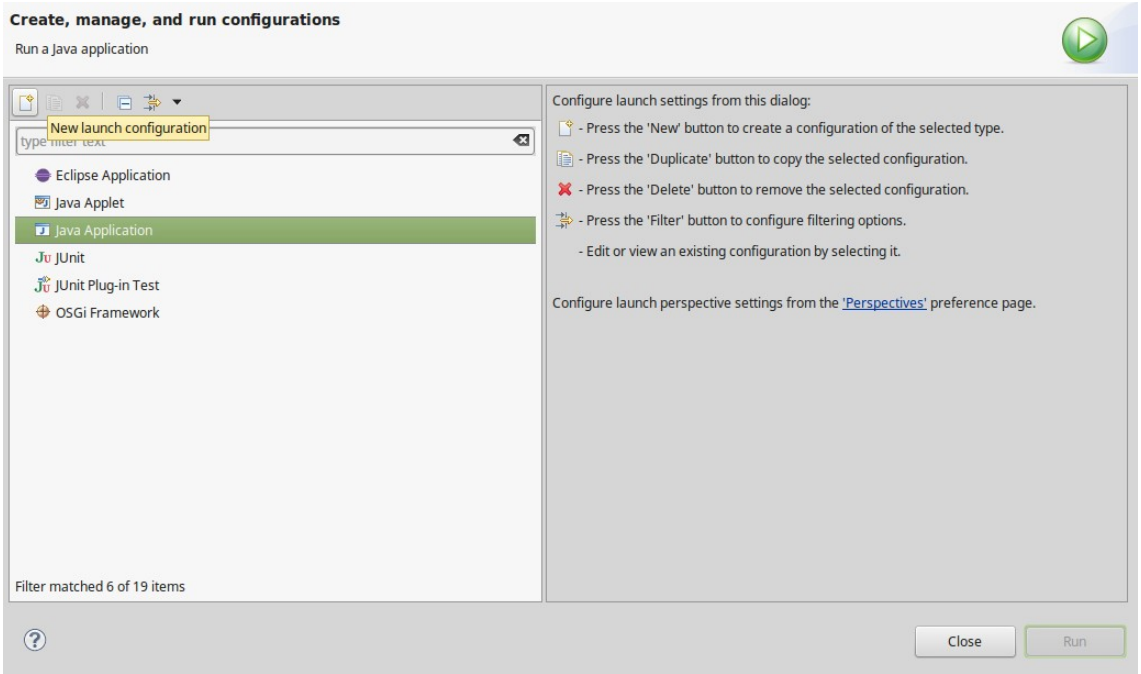

Fig. 36: Nueva configuración de lanzamiento

Dar un nombre a la aplicación (en la imagen el nombre es CrearAgentes), seleccionamos el proyecto (en la imagen MatrimonioEstable) y la clase principal, *main class*, que deberá ser *jade.Boot3*

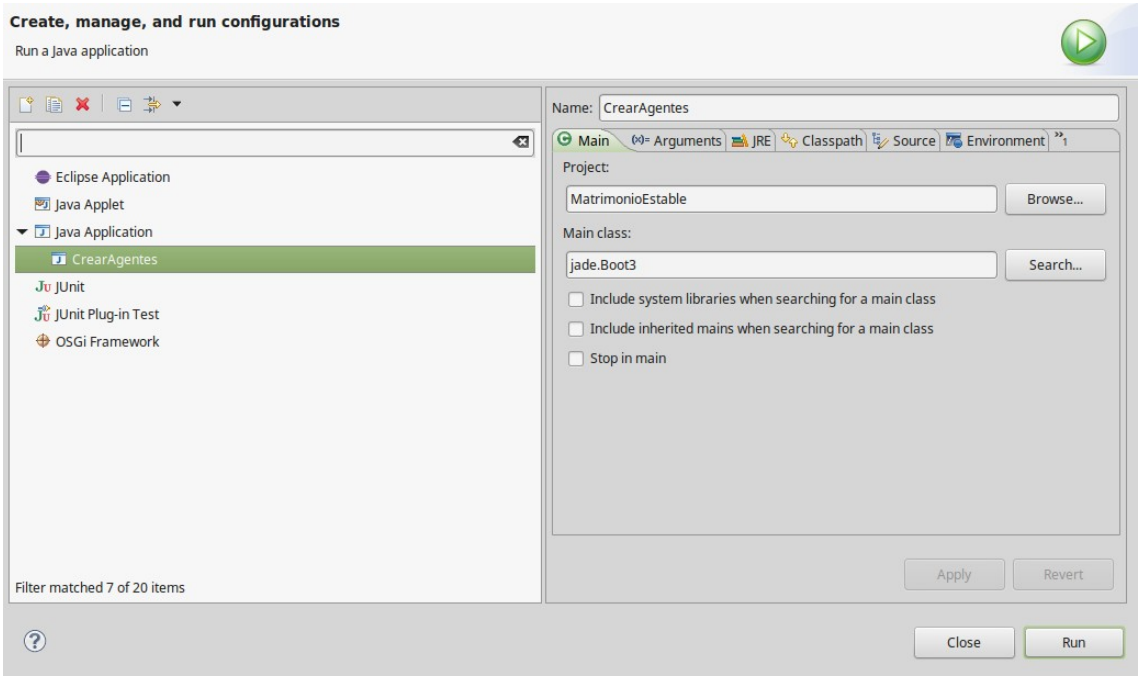

Fig. 37: Configuración de la main class

Pasar a la pestaña *Arguments* e introducir:

-jade_domain_df_maxresult 800 -gui gestor:tfgMarriageStable.GestorAgentes

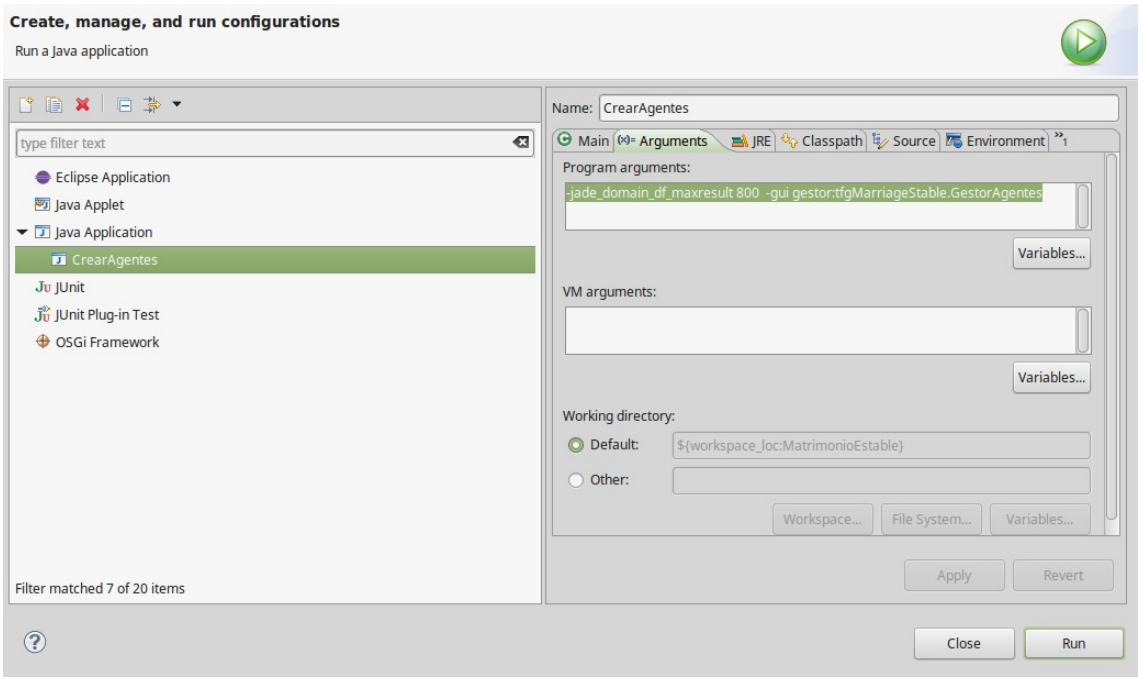

Fig. 38: Argumentos

Apretar el botón *Close.*

A partir de este momento, si se quiere ejecutar el programa principal no hay más que ir al botón de *Run As* y seleccionar la aplicación creada, en este caso CrearAgentes.

6.1.4.4 Ejecutar con opciones

Como se ha visto en la sección 6.1.3.3.1 Opciones avanzadas (añadir argumentos), el programa permite modificar su funcionamiento con una serie de parámetros.

Para introducir estos parámetros solo hay que seleccionar el menú Run/Run Configurations, marcar la aplicación deseada, en el ejemplo CrearAgentes y seleccionar la pestaña Arguments. Una vez allí se añaden, entre paréntesis y separados por espacios las opciones deseadas.

-jade_domain_df_maxresult 800 -gui gestor:tfgMarriageStable.GestorAgentes("newfile" "completo" 3)

En este caso se activan las opciones “newfile” “completo” y 3 iteraciones.

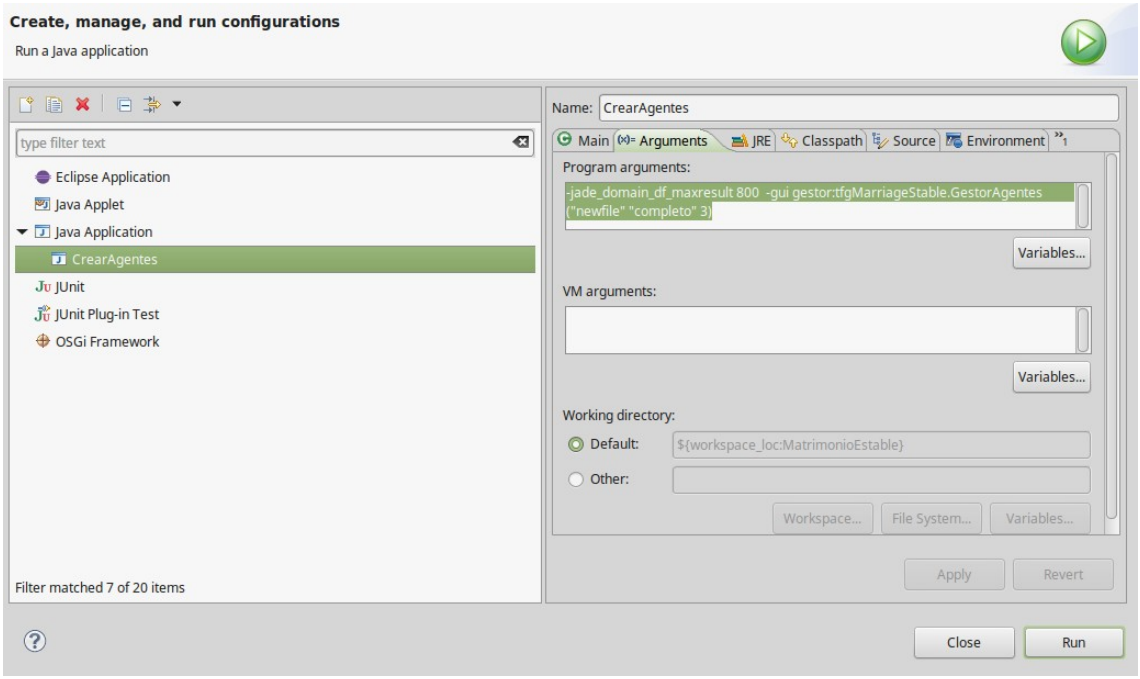

Fig. 39: Ejecución con opciones

6.1.4.5 Ejecución del clasificador por separado

Para ejecutar el clasificador sin necesidad de crear nuevas parejas, utilizando un archivo de entrenamiento y opcionalmente un archivo de test, se puede crear una nueva aplicación, tal y como se ha explicado en el apartado anterior o cambiar la configuración de la ya existente.

Lo único que hay que hacer será cambiar la clase de agente que se iniciará, iniciando el agente tfgMarriageStable.Weka y en los argumentos añadir los archivos que se quieren analizar.

-jade_domain_df_maxresult 800 -gui gestor:tfgMarriageStable.Weka("archivoEntrenamiento" "archivoTest")

6.1.4.6 Ejecución del paquete de pruebas

Para ejecutar el programa GaleShapleyUnitario[18] se procede, igual que en los casos anteriores, a crear una nueva aplicación. Sin embargo, en esta ocasión la *main class* será *pruebasGaleShapley.GaleShapleyUnitario*

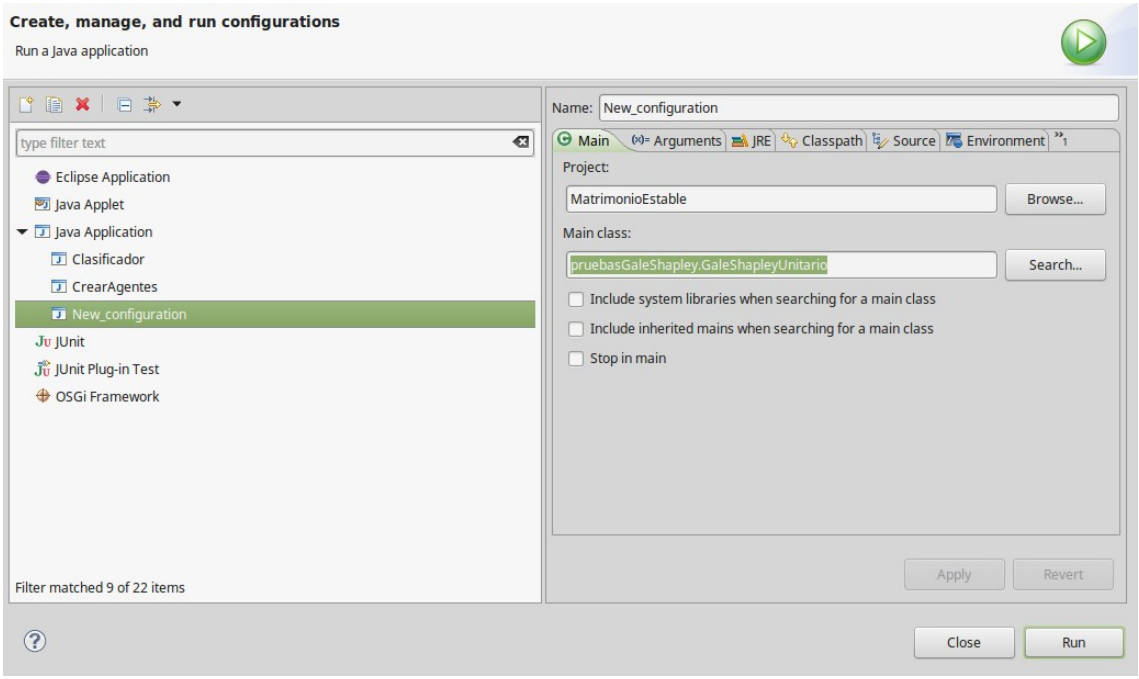

Fig. 40: GaleShapleyUnitario

En cuanto a los argumentos, se pueden dejar en blanco, o introducir dos archivos con elementos X, Y y sus listas de preferencias.

Para ejecutar el programa CheckParejasFiles[19] se procede , igual que en los casos anteriores a crear una nueva aplicación. Sin embargo, en esta ocasión la main class será pruebasGaleShapley.CheckParejasFiles

6.1.5 Archivos adicionales

Además de los archivos de las clases y los ya explicados, el programa utiliza dos archivos auxiliares más para guardar información

comunicaciones.csv Archivo que guarda los milisegundos que tardan los elementos en completar todas sus listas de favoritos y ordenarlas. Cada vez que un elemento completa esta lista envía un mensaje al gestor de agentes. Cuando ha recibido todos los mensajes calcula la diferencia de tiempo entre el momento de recibir el último y el primero y lo almacena junto con el número de parejas.

18 Ver sección 6.1.3.2.2 Paquete pruebasGaleShapley
19 Ver sección 6.1.3.2.2 Paquete pruebasGaleShapley

Ejemplo:

parejas,tiempo

400,8801

400,15990

400,15368

tiemposEmparejamientos.csv Almacena el tiempo que tarda en completarse el emparejado de los elementos una vez que se han recibido todas las listas de preferencias, así como en qué iteración se ha producido y si se han guardado todas las parejas no estables encontradas(Completo) o no (espacio en blanco)

Parejas,Tiempo,Iteración,Completo?

400,35800,1,completo

400,39505,1,completo

400,45443,2,completo

400,31597,3,completo

5,0,1,

5,0,2,

5,1,3,completo

5,0,4,completo

5,0,5,completo