MIARFID Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

ENTORNO DE NEGOCIACIÓN AUTOMÁTICA BILATERAL EN PYTHON

Autor: Pascual Andrés Carrasco Gómez
Asignatura: Sistemas multiagente (SMA)

Índice

ESTRUCTURA

El entorno de negociación automática bilateral está compuesto por los siguientes módulos:

- dominio.py: Permite definir un dominio mediante un rango de valores y la especificación de un tipo para cada valor.
- agente[1,2].xml: Define el conjunto de parámetros con los que va a negociar el agente [1,2] en la negociación.
- funciones_utilidad.py: Permite definir funciones de utilidad.
- estrategias_concesion.py: Permite definir funciones de concesión.
- estrategias_aceptacion.py: Permite definir funciones de aceptación.
- negociacion.py: Une todos los componentes anteriores permitiendo que los dos agentes negocien mediante el intercambio de ofertas de Rubinstein. Es el encargado de generar ofertas mediante un algoritmo genético y de visualizar los datos gráficamente y por terminal.

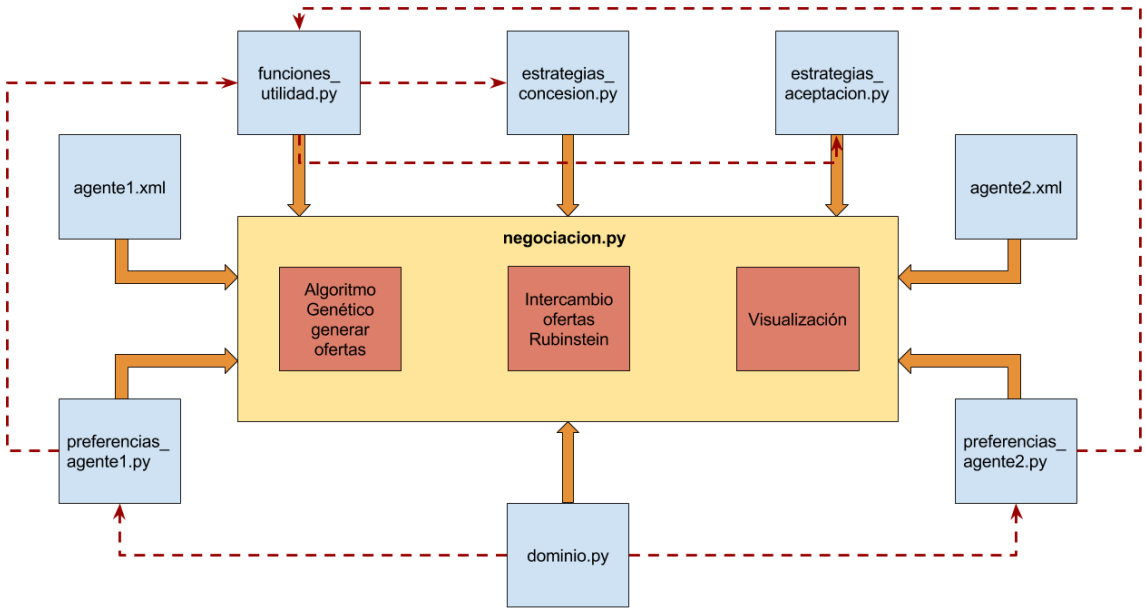


Imagen 1: Estructura de framework de negociación automática bilateral.

A continuación se comentan los módulos con más detalle indicando que parámetros y qué factores se han de tener en cuenta para el correcto funcionamiento del framework.

Módulos que puede configurar el usuario (Externos)

dominio.py

El framework permite trabajar con cualquier dominio que defina el usuario. Para la definición del dominio se necesitan dos parámetros, un parámetro para definir de qué tipos son los atributos del dominio y un segundo parámetro para definir qué rango de valores puede tomar cada atributo del dominio.
El dominio trabaja con tres tipos:

- int: Define un valor numérico entero.
- float: Define un valor numérico en coma flotante.
- list: Define una lista de valores.

El dominio requiere de un rango de valores para cada atributo con la finalidad de que el algoritmo genético pueda generar ofertas. Los rangos de valores son dependientes del tipo especificado para el atributo. Un atributo de tipo “int” o de tipo “float” tiene un rango de valores correspondiente a un número, entero o en coma flotante respectivamente, que permite definir el valor máximo que puede tomar ese atributo. Un atributo de tipo “list” contiene una lista de valores de cualquier tipo definido en python2.7.
Es importante tener en cuenta que el número de tipos declarados ha de coincidir con el número de rango de valores especificado.

preferencias_agente[1,2].py

En estos dos módulos se definen las preferencias que tienen los agentes [1,2] para una oferta recibida correspondiente al dominio. En la ilustración de la estructura del framework (Imagen 1) los módulos de preferencias de los agentes tienen una dependencia con el dominio del problema, esta dependencia a nivel de implementación no existe pero hace referencia a dos factores importantes que se han de tener en cuenta, el vector de pesos y el vector de valoraciones de una oferta han de tener el mismo número de elementos que los atributos definidos en el dominio.
El vector de pesos (w_agente) debe cumplir la restricción:

$$\sum_{i}^{N} w_{agente_i} = 1$$, siendo N = número de elementos definidos en el dominio

El vector de valoraciones debe especificar para cada atributo definido en el dominio qué valoración le aporta al usuario en un rango [0.0,...,1.0].

agente[1,2].xml

Los ficheros agente[1,2].xml permiten definir la configuración de negociación de cada agente, especificando mediante lenguaje de marcado los siguientes parámetros:

- nombre: nombre del agente.
- atributos: número de atributos definidos en el dominio.
- oferta->tipo: Tipo de oferta que queremos que devuelva el algoritmo genético.
  Tipos: 1,2 o 3. Se especificarán con detalle cuando se explique el algoritmo genético generador de ofertas en los apartados posteriores.

- futilidad->tipo: Función de utilidad que va a utilizar el agente en la negociación. El tipo (número) debe existir en el módulo “funciones_utilidad.py”.
- concesion->tipo: Método de concesión que va a utilizar el agente en la negociación. El tipo (número) debe existir en el módulo “estrategias_concesion.py”.
- concesion->parametros: Conjunto de parámetros que corresponden al argumento “parametros” definido en la cabecera de la función “actualizar_concesion” en el módulo “estrategias_concesion.py”.
- concesion->inicial: Es el valor de concesión inicial que tiene el agente al empezar la negociación. Este valor se define en el rango [0.0,...,1.0].
- aceptacion->tipo: Criterio de aceptación de oferta del agente en la negociación. El tipo (número) debe existir en el módulo “estrategias_aceptacion.py”.
- aceptacion->aceptar: Es el valor de concesión mínimo dispuesto a conceder por un agente. Este valor se define en el rango [0.0,...,1.0].
- tiempo: Es el tiempo de negociación que tiene el agente para llegar a un acuerdo. El tiempo se define en segundos.

funciones_utilidad.py

En este módulo se definen las funciones de utilidad. Una función de utilidad es una función matemática abstracta que encapsula las preferencias o gustos que tiene un agente. Las funciones de utilidad permiten al agente evaluar una oferta. Este módulo es dependiente del módulo preferencias_agente[1,2].py, ya que es necesario conocer las preferencias de un agente para definir su función de utilidad.
Por defecto se ha implementado una función de utilidad lineal, en el módulo también se ha implementado una función de utilidad denominada de tipo 2 para comprobar que la inserción de nuevas funciones de utilidad funciona correctamente.
Para añadir una nueva función de utilidad simplemente se ha de añadir una nueva opción dentro de la función “funcion_utilidad” e implementarla. La cabecera de la función de utilidad contiene los siguientes parámetros:

- agente: 1 = agente1 ; 2 = agente2.
- opcion: Opción para elegir la función de utilidad a utilizar.
- oferta: Oferta a la cual se le aplica la función de utilidad.

La función “funcion_utilidad” retorna la función de utilidad calculada para la oferta especificada como parámetro en su cabecera.

estrategias_concesion.py

La negociación es un proceso de concesión hacia un punto de acuerdo común, es decir, se empieza con unas aspiraciones las cuales se van reduciendo a medida que vemos que el acuerdo no se concreta. En este módulo se definen qué estrategias de concesión puede utilizar un agente en el framework. Se han implementado dos estrategias de concesión:

- Estrategia de concesión temporal: Se concede a medida que avanza el tiempo.
- Estrategia de concesión basada en el comportamiento: Se concede lo que se cree que ha concedido el otro agente. Se han implementado las tres variantes que se han estudiado en clase (relativo, absoluto, promediado).

Para añadir una nueva estrategia de concesión simplemente se ha de añadir una nueva opción dentro de la función “actualizar_concesion” e implementarla. La cabecera de la función “actualizar_concesion” contiene los siguientes parámetros:

- agente: 1 = agente1 ; 2 = agente2.
- tipo_fu: Función de utilidad que utiliza el agente.
- opcion: Opción para elegir la función de concesión a utilizar.
- T_agente: Tiempo de negociación del agente.
- ofertas_a: Lista con todas las ofertas emitidas por el agente.
- ofertas_b: Lista con todas las ofertas recibidas por el agente.
- parametros: Conjunto de parámetros necesarios para calcular la función de concesión. Es una lista de parámetros que coincide con los parámetros de concesión especificados en el fichero agente[1,2].xml.
- t_inicial: Tiempo de inicio de las negociaciones.

La función “actualizar_concesion” retorna el valor de concesión calculado según la estrategia seleccionada.

estrategias_aceptacion.py

En este módulo se definen los criterios de aceptación que puede utilizar un agente en el framework. Los criterios de aceptación definen cuando un agente acepta una oferta dando por concluida la negociación. Se han definido dos criterios de aceptación en este módulo:

- Criterio de aceptación básico: El agente acepta una oferta recibida si la función de utilidad de dicho agente para dicha oferta es mayor que el valor de concesión que tiene dicho agente en ese momento. El agente se retira de la negociación sin llegar a un acuerdo si su valor de concesión es menor que el valor de concesión mínimo que está dispuesto a llegar (s_aceptacion).
- Criterio de aceptación avanzado: El agente acepta una oferta recibida si la función de utilidad de dicho agente para dicha oferta es mayor que el valor de concesión que tiene dicho agente en ese momento. En el caso de que el valor de concesión del agente sea menor que su valor de concesión mínimo (s_aceptacion), pueden pasar dos cosas, se envía como respuesta la oferta con mayor función de utilidad del conjunto de ofertas recibidas siempre y cuando la función de utilidad de dicha oferta sea mayor que el valor de concesión mínimo (s_aceptación), si no se cumple lo anterior el agente se retira de la negociación sin llegar a un acuerdo.

Para añadir una nueva estrategia de aceptación simplemente se ha de añadir una nueva opción dentro de la función “aceptacion” e implementarla. La cabecera de la función “aceptacion” contiene los siguientes parámetros:

- agente: 1 = agente1 ; 2 = agente2
- opcion: Opción para elegir la función de aceptación a utilizar
- fu_oferta: Función de utilidad del agente
- s_actual: Concesión actual del agente
- s_aceptacion: Concesión mínima a la que está dispuesto el agente a llegar
- ofertas_a: Lista con todas las ofertas emitidas por el agente
- ofertas_b: Lista con todas las ofertas recibidas por el agente

La función “aceptacion” retorna una lista vacía si no se aplica ningún criterio de aceptación (o fin de negociación), si se cumple un criterio de aceptación el retorno de la función debe ser una lista de tres componentes:

return[0]: bool si aceptamos o no la oferta
return[1]: un mensaje que se mostrará por pantalla
return[2]: una oferta de contestación ; "" en caso de no devolver ninguna oferta

Módulo de negociación

El módulo de negociación (negociación.py) es el núcleo del framework y está formado por tres submódulos internos que se comentan con más detalle a continuación:

Protocolo de intercambio de ofertas Rubinstein (Regateo)

Este módulo es el encargado de interconectar todos los módulos externos estableciendo un intercambio de ofertas entre los dos agentes, corresponde a la función "negociacion" definida dentro del módulo negociacion.py. El protocolo de intercambio de ofertas de Rubinstein se divide en rondas, donde cada ronda está constituida de una oferta (o aceptación) por uno de los agentes y una contraoferta (o aceptación) del otro agente, teniendo en cuenta que en cualquier turno uno de los agentes puede abandonar la negociación. Las negociaciones no son infinitas ya que cada agente tiene un deadline privado para terminar la negociación ("tiempo" en el fichero agente[1,2].xml).

Algoritmo genético de generación de ofertas

La generación de ofertas en el framework se realiza mediante un algoritmo genético.
El algoritmo genético corresponde a la función "gen_ofertas_genetico" que se encuentra dentro del módulo "negociación.py" y cuya cabecera es la siguiente:

- agente: 1 = agente1 ; 2 = agente2
- ite: Número de iteraciones. (Generaciones del algoritmo genético).
- nip: Número de individuos por población.
- size_v: Número de elementos del vector v. (nº atributos de una oferta).
- s_max: Valor de concesión máximo.
- s_min: Valor de concesion mínimo.
- op: Opción para seleccionar la oferta a devolver (1: max; 2: min; 3: random).
- op_fu: Función de utilidad del agente que invoca la función.
- tipos_oferta: Tipos de datos que tiene una oferta (definidos en el dominio).
- rango_valores_oferta: Valores que puede tomar cada componente de la oferta (definidos en el dominio).

El algoritmo genético se describe con detalle a continuación, comentando todas sus fases.

Representación

Una oferta representa un individuo dentro de la población, y el fitness de un individuo es la función de utilidad (del agente que invoca el algoritmo, parámetro "op_fu") de una oferta.

Población inicial

El número de individuos de la población es constante en todas las generaciones del algoritmo, este número viene definido por el parámetro "nip" (nip = número individuos población). La población inicial se obtiene generando "nip" ofertas (individuos) de forma aleatoria.

Selección

El criterio de selección es obtener los "nip"/2 individuos de la población (padres) con mayor valor de función de utilidad.

Cruce: (Aleatorio por atributos)

El cruce se realiza para cada par de padres (el último padre se cruza con el primer padre), cada cruce permite obtener un hijo. El número de atributos de la oferta que se cruzan es un entero aleatorio y los atributos que se intercambian de la oferta también se obtienen de forma aleatoria. Ejemplo:

Oferta = 5 atributos de tipo "int", con rango de valores para todos los atributos de 0 a 9.

nip = 5

número de padres = 5/2 = 2

número de hijos = 2/2 = 1

padre1 = [5,2,3,9,7]

padre2 = [4,1,9,6,5]

número de atributos cruce (aleatorio) = 2

atributos a cruzar (aleatorio) = [3,1]

Se realiza el cruce con los atributos a cruzar [3,1] entre padre1 y padre2:

[5,2,3,9,7] [4,1,2,6,5] = [4,2,2,9,7]

hijo = [4,2,2,9,7]

Mutación: (Intercambio aleatorio de un elemento de la oferta)

La mutación se realiza individualmente para cada hijo. La mutación consiste en obtener un atributo de forma aleatoria e intercambiar su valor por un valor obtenido de forma aleatoria dentro del rango de valores que puede tomar ese atributo (definido en el dominio). Ejemplo:

Imaginemos que tenemos dos hijos obtenidos por la fase de cruce y las ofertas con las que trabajamos son las que hemos definido en el ejemplo de la fase de cruce.

hijo1 = [3,1,5,4,3]

hijo2 = [5,2,3,8,6]

Mutación de hijo1:

- atributo a mutar (aleatorio) = 4
- valor nuevo para atributo 4 (aleatorio) = 2
- hijo1 = [3,1,5,2,3]

Mutación de hijo2:

- atributo a mutar (aleatorio) = 1
- valor nuevo para atributo 1 (aleatorio) = 6
- hijo2 = [6,2,3,8,6]

Reemplazo

Se reemplazan los individuos de la población con menor valor de función de utilidad por los hijos obtenidos, es decir, si tenemos dos hijos se eliminarían los dos individuos de la población con menor valor de función de utilidad y se insertarán los dos hijos.

Generaciones (Iteraciones)

El algoritmo genético se detiene cuando realiza el número de iteraciones especificadas con el parámetro "ite".

El algoritmo genético trabaja con dos datos globales para todas las generaciones, el primer dato es una lista de “ofertas válidas” que actualiza en cada generación (con cada nueva población de individuos), las “ofertas válidas” son aquellas ofertas cuya función de utilidad se encuentra dentro del rango definido por “s_min” y “s_max” (pasados como parámetros en la función), el segundo dato es la “mejor oferta” obtenida hasta la fecha.
El algoritmo genético retorna una tupla formada por una oferta y su función de utilidad (la función de utilidad corresponde al agente que invoca la función mediante el parámetro “op_fu”). La oferta que retorna el algoritmo genético depende de dos factores:

- Si (número_ofertas_válidas = 0) retorna la “mejor oferta” obtenida hasta la fecha.
- Si no: retorna la oferta de ofertas válidas según el criterio indicado en el parámetro “op”.

El parámetro “op” puede tomar tres valores (1,2 y 3):

- 1: Mejor oferta del conjunto de ofertas válidas.
- 2: Peor oferta del conjunto de ofertas válidas.
- 3: Oferta aleatoria del conjunto de ofertas válidas.

Visualización

El módulo de visualización es el encargado de mostrar al usuario el proceso de negociación en tiempo real. El módulo de “visualización” se encuentra dentro del módulo “negociación.py”. La información que devuelve consiste en una gráfica donde el eje “x” es el rango [0...1] de función de utilidad del agente 1 y el eje “y” es el rango [0...1] de función de utilidad del agente 2. Los plots (x,y) se realizan de la siguiente forma:

- El agente 1 recibe una oferta del agente 2, el valor de “x” = función de utilidad del agente 1 sobre la oferta, el valor de “y” = función de utilidad del agente 2 sobre la oferta.
- El agente 2 recibe una oferta del agente 1, el valor de “x” = función de utilidad del agente 1 sobre la oferta, el valor de “y” = función de utilidad del agente 2 sobre la oferta.

Los valores de la gráfica se muestran en forma de tabla por la terminal para tener una visualización más completa de la negociación.
Al finalizar la negociación se muestra un mensaje por pantalla con el resultado obtenido:

- Agente: Agente que acepta la oferta.
- Criterio: Es el mensaje indicado en la definición de estrategia de aceptación.
- Oferta: Es la oferta aceptada.
- Función de utilidad: Es la función de utilidad de la oferta aceptada.
- Concesión: Concesión del agente en el momento que acepta la oferta.
- Número de ofertas generadas: Número de ofertas generadas que han hecho falta para llegar a un acuerdo (o no) en la negociación.

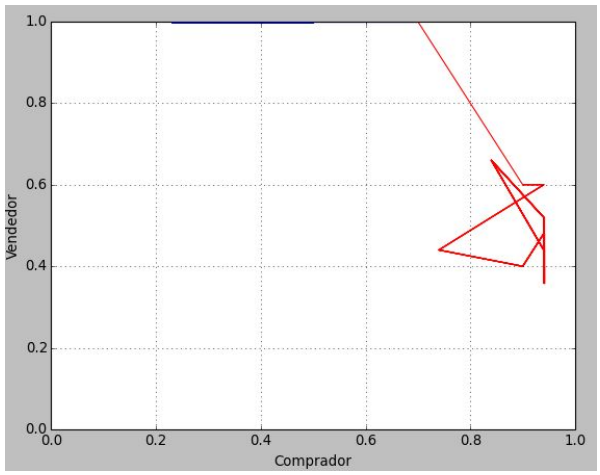


Imagen 2: Representación gráfica de la negociación.

```
##########################################################################################
Sentido (oferta)                f_utilidad_emisor       f_utilidad_receptor
##########################################################################################
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.94                    0.36
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.35
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.94                    0.4
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.5
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.94                    0.44
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.23
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.94                    0.48
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.5
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.94                    0.52
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.35
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.84                    0.66
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.43
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.94                    0.44
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.23
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.94                    0.48
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.5
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.9                     0.4
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.43
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.74                    0.44
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.31
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.94                    0.6
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.5
------------------------------------------------------------------------------------------
Comprador  -->  Vendedor                0.9                     0.6
------------------------------------------------------------------------------------------
Vendedor  -->  Comprador                1.0                     0.7
------------------------------------------------------------------------------------------
-----------------------------------
El agente 1 acepta la oferta
-----------------------------------
Criterio:  Se cumple fu_oferta > s_actual
Oferta:  [32, 1000, 14.0, 4, 1276.4926722787839, 'plata']
Funcion utilidad:  0.7
Concesion:  0.391304347826
Numero de ofertas generadas:  24
```

Imagen 3: Valores de la gráfica mostrados por consola.

Manual de usuario

En este manual de usuario se van a explicar los pasos detalladamente para que el framework de negociacion automatica bilateral funcione correctamente. Se van a generar el conjunto de ficheros de configuración con la finalidad de que el usuario sea capaz de utilizar la aplicación.

Dominio

Para crear un dominio debemos crear un fichero con el nombre “dominio.py”, en este fichero el usuario debe definir los tipos de los atributos con los que va a trabajar y el rango de valores que puede tomar cada atributo. Nuestro dominio ejemplo consiste en una negociación entre un vendedor y un comprador de portátiles, en concreto se definen los siguientes atributos con sus correspondiente tipo y rango de valores:

Atributo	Tipo	Rango de valores
RAM	list	2,4,8,16,32,64 (GB)
HDD	list	120,240,500,750,1000,2000 (GB)
Pulgadas	list	10.1,11.6,13.3,14.0,15.6,17.3
Peso	int	5
Precio	float	(RAM[len(RAM)-1]*HDD[len(HDD)-1])/100.0
Color	list	negro,azul,blanco,rojo,plata

Tabla 1: Definición del dominio de compra/venta de portatiles.

En el anexo de este documento se muestra la implementación del fichero “dominio.py” para el dominio descrito en este apartado.

Preferencias del agente 1 (Comprador)

Para definir las preferencias del agente 1 (comprador en nuestro caso) se debe crear un fichero con el nombre “preferencias_agente1.py”. Para nuestro ejemplo se han definido dos funciones, una función “w_agente1()” que define los pesos sobre los atributos que tiene el agente 1, son los que se muestran a continuación:

Atributo	RAM	HDD	Pulgadas	Peso	Precio	Color
Pesos	0.3	0.2	0.2	0.1	0.1	0.1

Tabla 2: Vector de pesos del agente 1.

Por otra parte se ha definido una función “v_agente1(oferta)” que devuelve el vector de valoraciones que tiene el agente 1 sobre una oferta, los valores que tiene el agente 1 sobre una oferta son los siguientes:

- Si la RAM es mayor o igual que 12 GB devuelve 1 como valoración a ese atributo, si la RAM se encuentra entre 6 y 12 GB devuelve un 0.5, por último, si la RAM es igual o menor que 6 GB devuelve un 0.1.
- Si el HDD es mayor o igual que 750 GB devuelve 1 como valoración a ese atributo, si el HDD se encuentra en 240 y 750 GB devuelve un 0.4, si el HDD es igual o menor que 240 GB devuelve un 0.
- Si las pulgadas de la pantalla son mayores que 15.6 pulgadas devuelve un 0.2 como valoración a ese atributo, si las pulgadas se encuentran entre 13.3 y 15.6 pulgadas (ambos incluidos) devuelve un 1, si tiene menor que 13.3 pulgadas devuelve un 0.
- Si el portátil pesa 4 kg o más devuelve 0 como valoración a este atributo, si pesa entre 2 y 4 kg devuelve un 0.4, si pesa igual o menos de 2kg devuelve 1.
- Si el precio del portatil es mayor o igual que 1000 euros devuelve un 0 como valoración a ese atributo, si el precio se encuentra entre 550.50 y 1000 euros devuelve un 0.4, si es igual o menor a 550.50 euros devuelve un 1.
- Si el portatil es de color negro devuelve un 1 como valoración a ese atributo, si es de color azul devuelve un 0.6, si es de cualquier otro color devuelve un 0.

En el anexo se muestra la implementación del fichero “preferencias_agente1.py”.

Preferencias del agente 2 (Vendedor)

Para definir las preferencias del agente 2 (vendedor en nuestro caso) se debe crear un fichero con el nombre “preferencias_agente2.py”. Para nuestro ejemplo se han definido dos funciones, una función “w_agente2()” que define los pesos sobre los atributos que tiene el agente 2, son los que se muestran a continuación:

Atributo	RAM	HDD	Pulgadas	Peso	Precio	Color
Pesos	0.2	0.1	0.1	0.2	0.3	0.1

Tabla 3: Vector de pesos del agente 2.

Por otra parte se ha definido una función “v_agente2(oferta)” que devuelve el vector de valoraciones que tiene el agente 2 sobre una oferta, los valores que tiene el agente 2 sobre una oferta son los siguientes:

- Si la RAM es mayor o igual que 32 GB devuelve 0.2 como valoración a ese atributo, si la RAM se encuentra entre 12 y 32 GB devuelve un 0.4, por último, si la RAM es igual o menor que 12 GB devuelve un 0.1.
- Si el HDD es mayor o igual que 1000 GB devuelve 0.2 como valoración a ese atributo, si el HDD se encuentra entre 500 y 1000 GB devuelve un 0.6, si el HDD es igual o menor que 500 GB devuelve un 1.
- Si las pulgadas de la pantalla son mayores o igual que 15.6 pulgadas devuelve un 0.6 como valoración a ese atributo, si las pulgadas se encuentran entre 13.3 y 15.6 pulgadas devuelve un 1, si tiene igual o menor que 13.3 pulgadas devuelve un 0.2.

- Si el portátil pesa 3 kg o más devuelve 1 como valoración a este atributo, si pesa entre 1 y 3 kg devuelve un 0.8, si pesa igual o menos de 1kg devuelve 0.2.
- Si el precio del portatil es mayor o igual que 1200 euros devuelve un 1 como valoración a ese atributo, si el precio se encuentra entre 600.75 y 1200 euros devuelve un 0.4, si es igual o menor a 600.75 euros devuelve un 0.2.
- Si el portatil es de color plata devuelve un 1 como valoración a ese atributo, si es de color azul devuelve un 0.6, si es de color negro devuelve un 0.4, si es de cualquier otro color devuelve un 0.1.

En el anexo se muestra la implementación del fichero “preferencias_agente2.py”.

Funciones de utilidad

Las funciones implementadas en los ficheros de preferencias de usuario se utilizan en el fichero de funciones de utilidad, este fichero se debe llamar “funciones_utilidad.py”. En este fichero debe estar definida la función con cabecera:

“funcion_utilidad(agente,opcion,oferta)”

donde las funciones de utilidad se escogen según la opción indicada. Se pueden añadir más funciones de utilidad simplemente definiendolas con una opción nueva.
En el anexo se muestra la implementación del fichero “funciones_utilidad.py”.

Estrategias de concesión

Las funciones implementadas en el fichero de funciones de utilidad se utilizan en el fichero de estrategias de concesion, este fichero se debe llamar “estrategias_concesion.py”. En este fichero debe estar definida la función con cabecera:

“actualizar_concesion(agente,tipo_fu,opcion,T_agente,ofertas_a,ofertas_b,parametros,t_inicial)”

donde las estrategias de concesión se escogen según la opción indicada. Se han definido las estrategias de concesión temporal y por comportamiento. Se pueden añadir más estrategias de concesión simplemente definiendolas con una opción nueva.
En el anexo se muestra la implementación del fichero “estrategias_concesion.py”.

Criterios de aceptación

Este fichero se debe llamar “estrategias_aceptacion.py”. En este fichero debe estar definida la función con cabecera:

“aceptacion(agente,opcion,fu_oferta,s_actual,s_aceptacion,ofertas_a,ofertas_b)”

donde las estrategias de aceptación se escogen según la opción indicada. Se han definido dos estrategias de aceptación una básica y una más avanzada. Se pueden añadir más estrategias de aceptación simplemente definiendolas con una opción nueva.
En el anexo se muestra la implementación del fichero “estrategias_aceptacion.py”.

Configuración de parámetros de cada agente

Deben existir dos ficheros de parámetros, uno para el agente 1 con el nombre agente1.xml y otro para el agente 2 con el nombre de agente2.xml.
En el anexo se muestra la implementación para la configuración del agente 1 y agente 2, pero estos dos ficheros se verán más a fondo en el documento de evaluación del framework.

Evaluación

El framework de negociación automática bilateral se ejecuta de la siguiente forma:

python2.7 negociacion,py

En este apartado se han realizado un conjunto de ejecuciones para evaluar el framework, para cada ejecución se muestra la configuración de parámetros de los ficheros agente1.xml y agente2.xml, la gráfica resultante de la negociación y el acuerdo (o no acuerdo) de la negociación que muestra el framework en el terminal.

Los resultados de las ejecuciones no son deterministas ya que las ofertas se generan con un algoritmo genético donde genera ofertas diferentes en cada ejecución.

El dominio con el que se trabaja en la evaluación es el ejemplo definido en el manual de usuario que consiste en una negociación para comprar/vender un portátil.

Se han escogido parámetros por defecto para realizar la evaluación, estos parámetros son los mismos para los dos agentes, se comentan a continuación:

- En el algoritmo genético se devuelve la mejor oferta encontrada (opción 1).
- La función de utilidad es lineal (opción 1)
- El criterio de aceptación es el avanzado (opción 2).
- La concesión inicial es 1.
- La concesión mínima que está dispuesto a llegar el agente es 0.3.
- El deadline de la negociación es de 300s (5 minutos).

Se han realizado ejecuciones variando las estrategias de concesión temporal y por comportamiento. Todos los datos obtenidos se adjuntan en la práctica, se encuentran almacenados dentro del directorio evaluación, son ficheros .txt con los resultados que imprime el framework por pantalla.

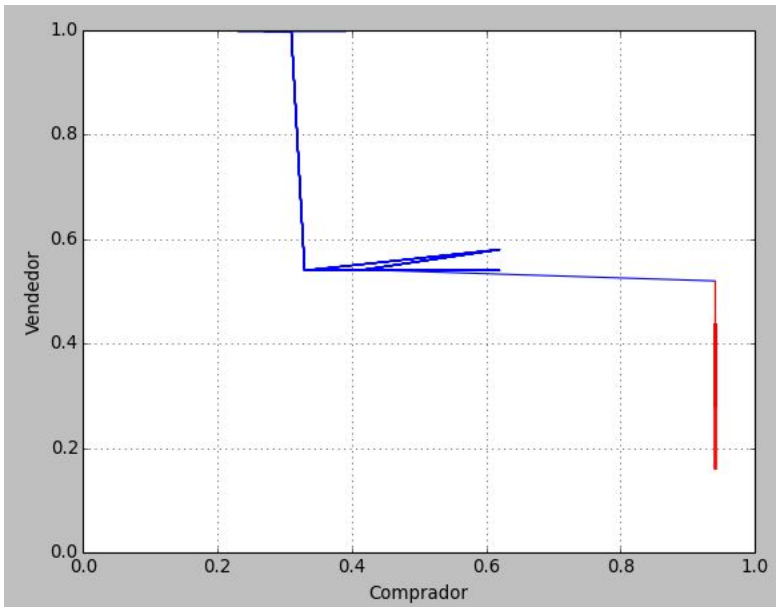
Boulware VS Conceder

agente1.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Comprador</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>1</tipo>
    <parametros>0.1 0.3</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

agente2.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Vendedor</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>1</tipo>
    <parametros>0.1 10</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```



El agente 2 acepta la oferta

Criterio: Se cumple fu_oferta > s_actual
Oferta: [16, 1000, 14.0, 2, 844.6225824605231, 'negro']
Funcion utilidad: 0.52
Concesion: 0.454702654523
Numero de ofertas generadas: 33

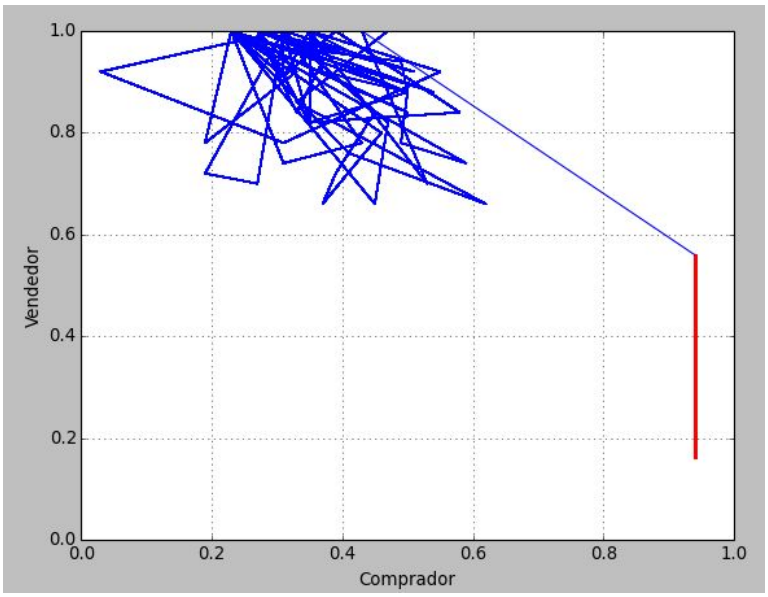
Boulware VS Comportamiento (Relativo)

agente1.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Comprador</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>1</tipo>
    <parametros>0.1 0.3</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

agente2.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Vendedor</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.3 3 Relativo</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```



El agente 2 acepta la oferta

Criterio: Se cumple fu_oferta > s_actual

Oferta: [16, 750, 14.0, 2, 802.888234340305, 'negro']

Funcion utilidad: 0.56

Concesion: 0.55

Numero de ofertas generadas: 219

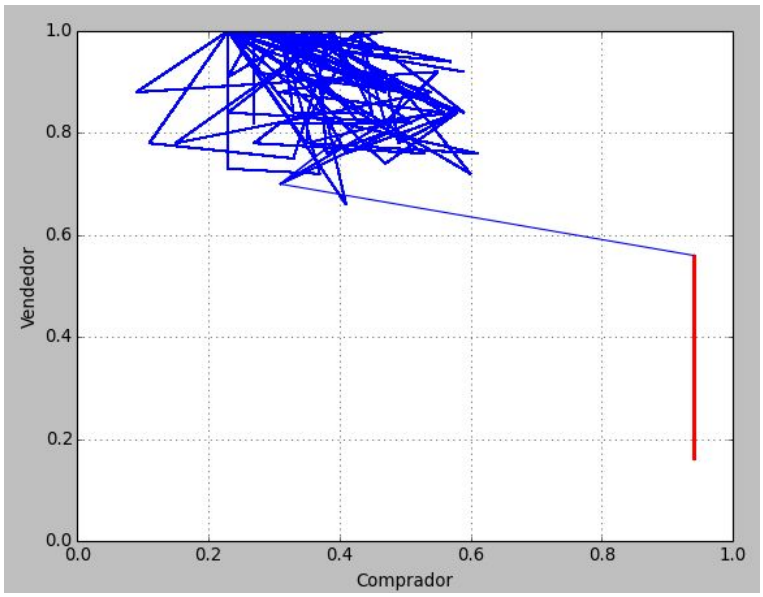
Boulware VS Comportamiento (Absoluto)

agente1.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Comprador</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>1</tipo>
    <parametros>0.1 0.3</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

agente2.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Vendedor</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.3 3 Absoluto</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```



El agente 2 acepta la oferta

Criterio: Se cumple fu_oferta > s_actual

Oferta: [16, 750, 14.0, 2, 915.2109480579134, 'negro']

Funcion utilidad: 0.56

Concesion: 0.55

Numero de ofertas generadas: 355

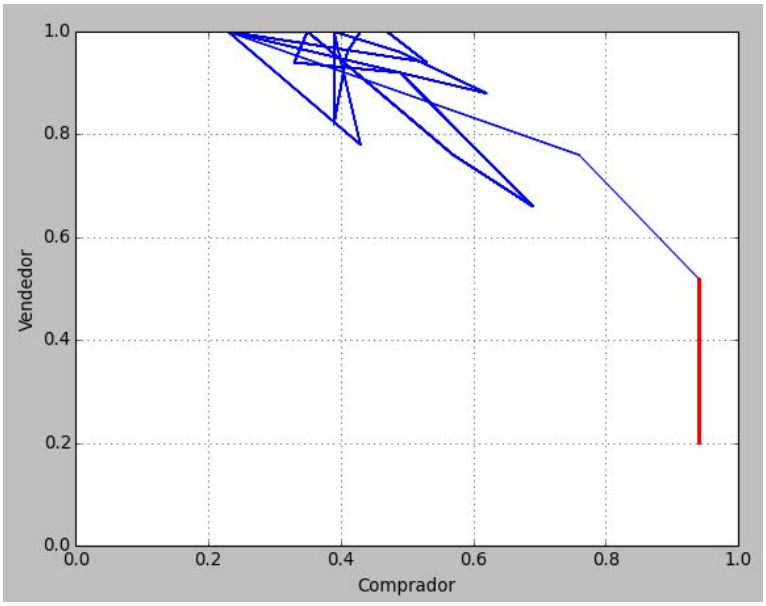
Boulware VS Comportamiento (Promediado)

agente1.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Comprador</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>1</tipo>
    <parametros>0.1 0.3</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

agente2.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Vendedor</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.3 3 Promediado</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```



El agente 2 acepta la oferta

Criterio: Se cumple fu_oferta > s_actual
Oferta: [16, 750, 15.6, 2, 808.6876546969346, 'negro']
Funcion utilidad: 0.52
Concesion: 0.506666666667
Numero de ofertas generadas: 67

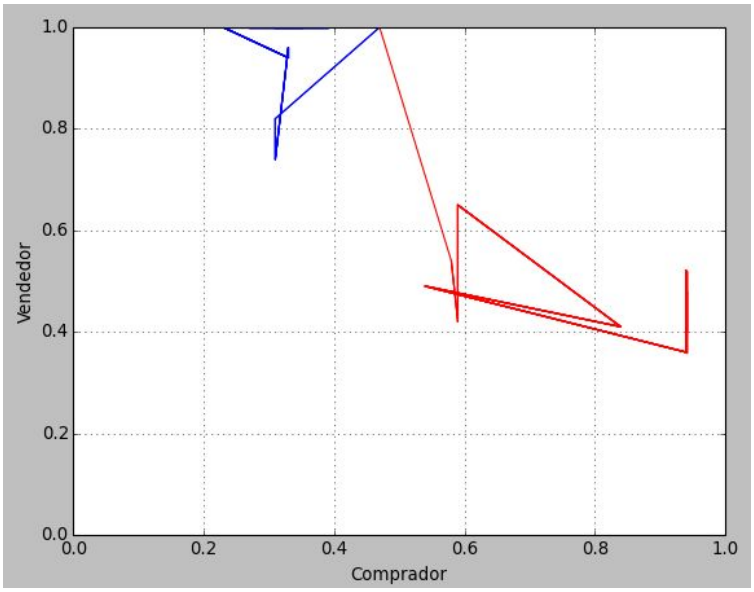
Conceder VS Comportamiento (Relativo)

agente1.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Comprador</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>1</tipo>
    <parametros>0.1 10</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

agente2.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Vendedor</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.3 3 Relativo</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```



El agente 1 acepta la oferta

Criterio: Se cumple fu_oferta > s_actual

Oferta: [8, 500, 14.0, 3, 1221.8818952123656, 'plata']

Funcion utilidad: 0.47

Concesion: 0.454702654523

Numero de ofertas generadas: 22

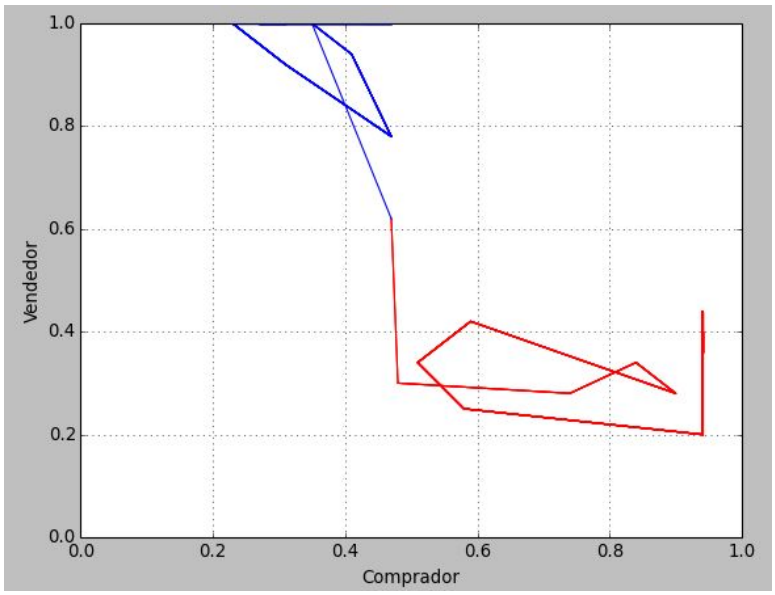
Conceder VS Comportamiento (Absoluto)

agente1.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Comprador</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>1</tipo>
    <parametros>0.1 10</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

agente2.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Vendedor</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.3 3 Absoluto</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```



El agente 2 acepta la oferta

Criterio: Se cumple fu_oferta > s_actual
Oferta: [4, 1000, 17.3, 2, 948.7810416554108, 'azul']
Funcion utilidad: 0.62
Concesion: 0.527777777778
Numero de ofertas generadas: 27

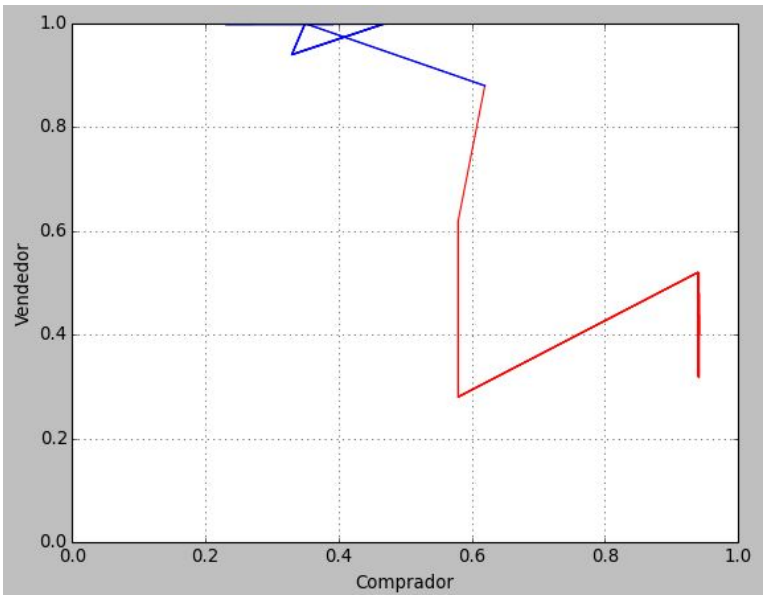
Conceder VS Comportamiento (Promediado)

agente1.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Comprador</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>1</tipo>
    <parametros>0.1 10</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

agente2.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Vendedor</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.3 3 Promediado</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```



El agente 1 acepta la oferta

Criterio: Se cumple fu_oferta > s_actual
Oferta: [16, 500, 14.0, 3, 1234.1651212536608, 'plata']
Funcion utilidad: 0.62
Concesion: 0.491219586473
Numero de ofertas generadas: 16

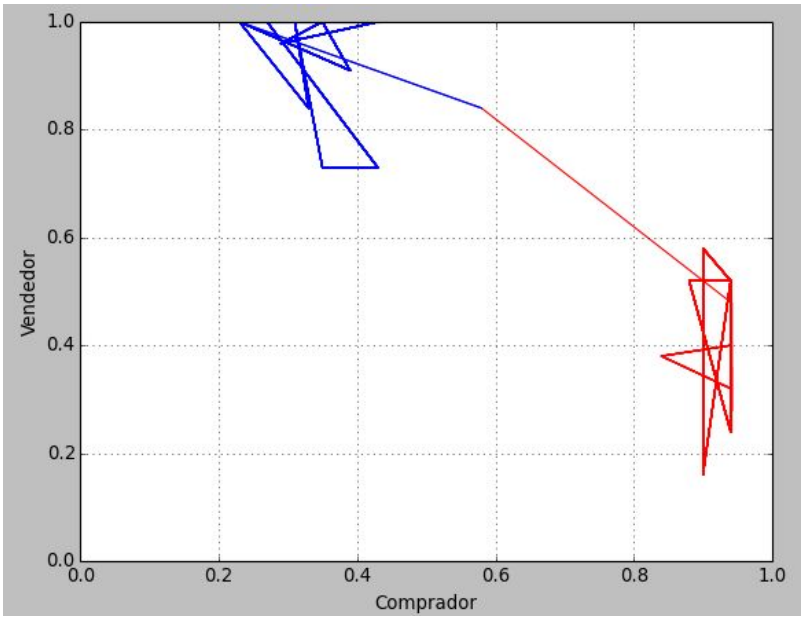
Comportamiento (Relativo) VS Comportamiento (Promediado)

agente1.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Comprador</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.3 3 Relativo</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

agente2.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Vendedor</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.3 3 Promediado</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```



El agente 1 acepta la oferta

Criterio: Se cumple fu_oferta > s_actual
Oferta: [64, 500, 14.0, 4, 1219.1736445419665, 'plata']
Funcion utilidad: 0.58
Concesion: 0.512727272727
Numero de ofertas generadas: 40

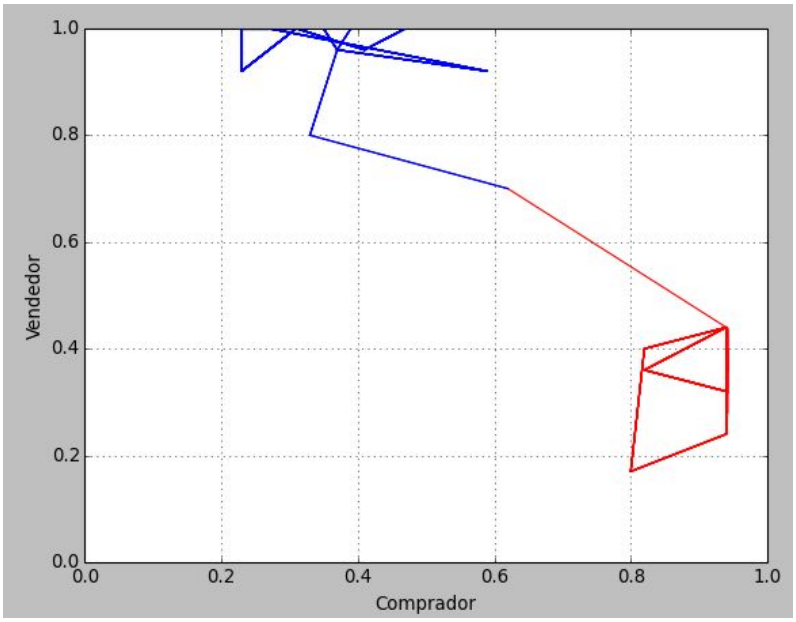
Comportamiento (Relativo) VS Comportamiento (Absoluto)

agente1.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Comprador</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.3 3 Relativo</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

agente2.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Vendedor</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.3 3 Absoluto</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

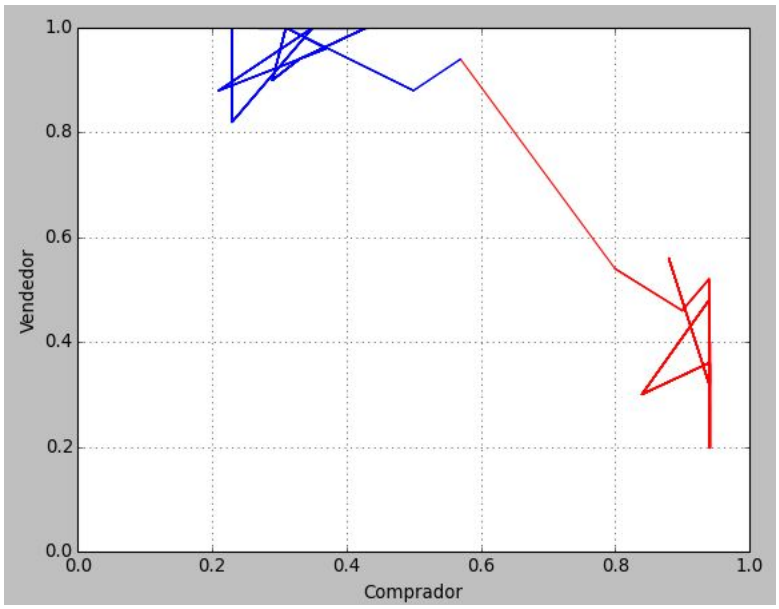


El agente 1 acepta la oferta

Criterio: Se cumple fu_oferta > s_actual
Oferta: [16, 500, 14.0, 5, 646.631277307632, 'plata']
Funcion utilidad: 0.62
Concesion: 0.566984126984
Numero de ofertas generadas: 34

Comportamiento (Promediado) VS Comportamiento (Absoluto)

agente1.xml	agente2.xml
<?xml version="1.0"?> <agente> <nombre>Comprador</nombre> <atributos>6</atributos> <oferta> <tipo>1</tipo> </oferta> <futilidad> <tipo>1</tipo> </futilidad> <concesion> <tipo>2</tipo> <parametros>0.3 3 Promediado</parametros> <inicial>1</inicial> </concesion> <aceptacion> <tipo>2</tipo> <aceptar>0.3</aceptar> </aceptacion> <tiempo>300</tiempo> </agente>	<?xml version="1.0"?> <agente> <nombre>Vendedor</nombre> <atributos>6</atributos> <oferta> <tipo>1</tipo> </oferta> <futilidad> <tipo>1</tipo> </futilidad> <concesion> <tipo>2</tipo> <parametros>0.3 3 Absoluto</parametros> <inicial>1</inicial> </concesion> <aceptacion> <tipo>2</tipo> <aceptar>0.3</aceptar> </aceptacion> <tiempo>300</tiempo> </agente>



El agente 1 acepta la oferta

Criterio: Se cumple fu_oferta > s_actual

Oferta: [8, 500, 14.0, 3, 1244.7366606285648, 'negro']

Funcion utilidad: 0.57

Concesion: 0.498550724638

Numero de ofertas generadas: 30

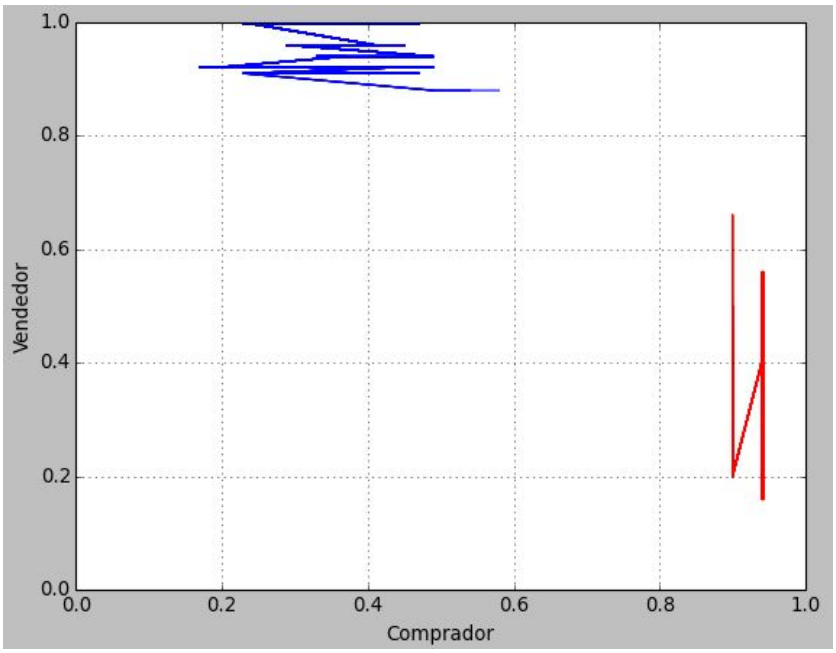
Boulware VS Boulware

agente1.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Comprador</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>1</tipo>
    <parametros>0.8 0.3</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

agente2.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Vendedor</nombre>
  <atributos>6</atributos>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>1</tipo>
    <parametros>0.8 0.3</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```



Se ha agotado el tiempo del agente 1 (Deadline = 300 s)
Numero de ofertas generadas: 892

Anexo

dominio.py

```
################################################################
# Autor: Pascual Andres Carrasco Gomez
# Asignatura: Sistemas multiagente (SMA)
# Trabajo: Entorno de negociacion automatica bilateral
# Descripcion: Definicion del dominio del problema
# Dominio: Compra/Venta de portatil
# Lenguaje: python2.7
################################################################

# Definicion del dominio
def dominio():

  # Definimos los tipos de atributos
  tipos = ["list","list","list","int","float","list"]

  # Definimos los valores que pueden tomar los atributos dentro del dominio
  RAM = [2,4,8,16,32,64]
  HDD = [120,240,500,750,1000,2000]
  pulgadas = [10.1,11.6,13.3,14.0,15.6,17.3]
  peso = 5 # Cota superior
  precio = (RAM[len(RAM)-1]*HDD[len(HDD)-1])/100.0 # Cota superior
  color = ["negro","azul","blanco","rojo","plata"]

  # Devolvemos el dominio
  return [tipos,[RAM,HDD,pulgadas,peso,precio,color]]
```

preferencias_agente1.py

```
################################################################
# Autor: Pascual Andres Carrasco Gomez
# Asignatura: Sistemas multiagente (SMA)
# Trabajo: Entorno de negociacion automatica bilateral
# Descripcion: Preferencias del agente 1 en la negociacion
# Dominio: Compra/Venta de portatil
# Lenguaje: python2.7
################################################################

#--------------------------------------
# Agente comprador
#--------------------------------------

# Devuelve el vector de pesos
def w_agente1():
  # Definimos los pesos
  return [0.3,0.2,0.2,0.1,0.1,0.1]
```

```
# Devuelve el vector de valoraciones para una oferta
def v_agente1(oferta):
  # Definimos vector de valoraciones
  v = [0]*len(w_agente1())
  # Valoracion atributo 1 = RAM (GB)
  if(oferta[0] >= 12):
    v[0] = 1
  elif(6 < oferta[0] < 12):
    v[0] = 0.5
  else:
    v[0] = 0.1
  # Valoracion atributo 2 = HDD (GB)
  if(oferta[1] >= 750):
    v[1] = 1
  elif(240 < oferta[1] < 750):
    v[1] = 0.4
  else:
    v[1] = 0
  # Valoracion atributo 3 = Pulgadas Monitor (Pulgadas)
  if(oferta[2] > 15.6):
    v[2] = 0.2
  elif(13.3 <= oferta[2] <= 15.6):
    v[2] = 1
  else:
    v[2] = 0
  # Valoracion atributo 4 = Peso (Kg)
  if(oferta[3] >= 4:
    v[3] = 0
  elif(2 < oferta[3] < 4):
    v[3] = 0.4
  else:
    v[3] = 1
  # Valoracion atributo 5 = Precio
  if(oferta[4] >= 1000):
    v[4] = 0
  elif(550.50 < oferta[4] < 1000):
    v[4] = 0.4
  else:
    v[3] = 1
  # Valoracion atributo 6 = Color
  if(oferta[5] == "negro"):
    v[5] = 1
  elif(oferta[5] == "azul"):
    v[5] = 0.6
  else:
    v[5] = 0
  # Retornamos el vector de valores
  return v
```

preferencias_agente2.py

```
#################################################################
# Autor: Pascual Andres Carrasco Gomez
# Asignatura: Sistemas multiagente (SMA)
# Trabajo: Entorno de negociacion automatica bilateral
# Descripcion: Preferencias del agente 2 en la negociacion
# Dominio: Compra/Venta de portatil
# Lenguaje: python2.7
#################################################################

#---------------------------------------
# Agente vendedor
#---------------------------------------

# Devuelve el vector de pesos
def w_agente2():
  # Definimos los pesos
  return [0.2,0.1,0.1,0.2,0.3,0.1]

# Devuelve el vector de valoraciones para una oferta
def v_agente2(oferta):
  # Definimos vector de valoraciones
  v = [0]*len(w_agente2())
  # Valoracion atributo 1 = RAM (GB)
  if(oferta[0] >= 32):
    v[0] = 0.2
  elif(12 < oferta[0] < 32):
    v[0] = 0.4
  else:
    v[0] = 1
  # Valoracion atributo 2 = HDD (GB)
  if(oferta[1] >= 1000):
    v[1] = 0.2
  elif(500 < oferta[1] < 1000):
    v[1] = 0.6
  else:
    v[1] = 1
  # Valoracion atributo 3 = Pulgadas Monitor (Pulgadas)
  if(oferta[2] >= 15.6):
    v[2] = 0.6
  elif(13.3 < oferta[2] < 15.6):
    v[2] = 1
  else:
    v[2] = 0.2
  # Valoracion atributo 4 = Peso (Kg)
  if(oferta[3] >= 3):
    v[3] = 1
  elif(1 < oferta[3] < 3):
    v[3] = 0.8
  else:
    v[3] = 0.2
  # Valoracion atributo 5 = Precio
```

```
 if(oferta[4] >= 1200):
   v[4] = 1
 elif(600.75 < oferta[4] < 1200):
   v[4] = 0.4
 else:
   v[3] = 0.2
 # Valoracion atributo 6 = Color
 if(oferta[5] == "plata"):
   v[5] = 1
 elif(oferta[5] == "azul"):
   v[5] = 0.6
 elif(oferta[5] == "negro"):
   v[5] = 0.4
 else:
   v[5] = 0.1
 # Retornamos el vector de valores
 return v
```

funciones_utilidad.py

```
import math
from preferencias_agente1 import * # w_agente1(), v_agente1(oferta)
from preferencias_agente2 import * # w_agente2(), v_agente2(oferta)

############################################################
# Autor: Pascual Andres Carrasco Gomez
# Asignatura: Sistemas multiagente (SMA)
# Trabajo: Entorno de negociacion automatica bilateral
# Descripcion: Modulo con las funciones de utilidad
# Lenguaje: python2.7
############################################################

# GESTIONAR FUNCIONES DE UTILIDAD
# Para introducir una funcion de utilidad nueva insertar una opcion nueva
# agente: 1 = Agente1 ; 2 = Agente2
# opcion: opcion para elegir la funcion de utilidad a utilizar
# oferta: oferta a la cual se le aplica la funcion de utilidad
def funcion_utilidad(agente,opcion,oferta):
 if(agente == 1):
   w = w_agente1()
   v = v_agente1(oferta)
 else: # agente == 2
   w = w_agente2()
   v = v_agente2(oferta)
 if(opcion == 1):
   #-----------------------------------------------------------
   # FUNCION DE UTILIDAD LINEAL
   #-----------------------------------------------------------
   aux = 0
   for i in range(0,len(w)):
     aux += w[i]*v[i]
   return aux
 elif(opcion == 2):
```

```
    #---------------------------------------------------
    # FUNCION DE UTILIDAD TIPO 2 (Solo para probar el modulo)
    #---------------------------------------------------
    aux = 0
    for i in range(0,len(w)):
      aux += (w[i]*(v[i]/2.0))+0.05
    return aux
  # elif(opcion == 3):
  # elif(opcion == 4):
  # elif(opcion == 5):
  # ...
  return 0
```

estrategias_concesion.py

```
import math
from datetime import datetime
from funciones_utilidad import *

############################################################
# Autor: Pascual Andres Carrasco Gomez
# Asignatura: Sistemas multiagente (SMA)
# Trabajo: Entorno de negociacion automatica bilateral
# Descripcion: Modulo con las estrategias de concesion
# Lenguaje: python2.7
############################################################

# GESTIONAR FUNCIONES DE CONCESION
# Para introducir una funcion de concesion nueva insertar una opcion nueva
# agente: 1 = Agente1 ; 2 = Agente2
# tipo_fu: Funcion de utilidad que utiliza el agente
# opcion: opcion para elegir la funcion de concesion a utilizar
# T_agente: Tiempo de negociacion del agente
# ofertas_a: lista con todas las ofertas emitidas por el agente
# ofertas_b: lista con todas las ofertas recibidas por el agente
# parametros: conjunto de parametros necesarios para calcular la funcion de concesion
# t_inicial: Tiempo de inicio de las negociaciones
# NOTA: Los parametros son los valores introducidos en parametros de concesion del xml del agente
def actualizar_concesion(agente,tipo_fu,opcion,T_agente,ofertas_a,ofertas_b,parametros,t_inicial):
  if(opcion == 1):
    #---------------------------------------------------
    # TEMPORAL
    #---------------------------------------------------
    #  0 < B < 1 = Boulware
    #  1 < B < inf = Conceder
    #---------------------------------------------------
    # Parametros
    RU = float(parametros[0])
    B = float(parametros[1])
    exponente = 1.0/B
    t_actual = datetime.now()
    t = t_actual - t_inicial
    t = float(t.seconds)
```

```
  s = 1 - ((1 - RU)*(math.pow((t/T_agente),exponente)))
  return s
elif(opcion == 2):
  #---------------------------------------------------------
  # COMPORTAMIENTO
  #---------------------------------------------------------
  # Basada en el comportamiento (a agente que llama a la funcion)
  #                                    (b agente destinatario)
  #---------------------------------------------------------
  # Parametros
  RU = float(parametros[0])
  D = int(parametros[1])
  tipo = parametros[2]
  if(len(ofertas_b) >= D and len(ofertas_a) >= 1): # Concesion por comportamiento
    s = 0
    if(agente == 1):
      # Obtenemos los valores w_agente y v_agente
      from preferencias_agente1 import *
      fu_oferta_ab_t_1 = funcion_utilidad(1,tipo_fu,ofertas_a[len(ofertas_a)-1])
      fu_oferta_ba_t_1 = funcion_utilidad(1,tipo_fu,ofertas_b[len(ofertas_b)-1])
      fu_oferta_ba_t_D = funcion_utilidad(1,tipo_fu,ofertas_b[len(ofertas_b)-D])
      if(D == 1):
        return 1
      else:
        fu_oferta_ba_t_D_1 = funcion_utilidad(1,tipo_fu,ofertas_b[len(ofertas_b)-D+1])
    else: # agente 2
      # Obtenemos los valores w_agente y v_agente
      from preferencias_agente2 import *
      fu_oferta_ab_t_1 = funcion_utilidad(2,tipo_fu,ofertas_a[len(ofertas_a)-1])
      fu_oferta_ba_t_1 = funcion_utilidad(2,tipo_fu,ofertas_b[len(ofertas_b)-1])
      fu_oferta_ba_t_D = funcion_utilidad(2,tipo_fu,ofertas_b[len(ofertas_b)-D])
      if(D == 1):
        return 1
      else:
        fu_oferta_ba_t_D_1 = funcion_utilidad(2,tipo_fu,ofertas_b[len(ofertas_b)-D+1])
    # tipo: relativo, absoluto, promediado
    if(tipo == "relativo"):
      s = min(1,max(RU,((1-fu_oferta_ba_t_D_1)/(1-fu_oferta_ba_t_D))*fu_oferta_ab_t_1))
    elif(tipo == "absoluto"):
      s = min(1,max(RU,fu_oferta_ab_t_1-fu_oferta_ba_t_D_1-fu_oferta_ba_t_D))
    else: # tipo = promediado
      s = min(1,max(RU,((1-fu_oferta_ba_t_1)/(1-fu_oferta_ba_t_D))*fu_oferta_ab_t_1))
    return s
  else:
    return 1
# elif(opcion == 3):
# elif(opcion == 4):
# elif(opcion == 5):
# ...
return 0
```

agente1.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Comprador</nombre>
  <atributos>6</atributos>
  <preferencias>
    <fichero>preferencias_agente1.py</fichero>
  </preferencias>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.1 3 promediado</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```

agente2.xml

```
<?xml version="1.0"?>
<agente>
  <nombre>Vendedor</nombre>
  <atributos>6</atributos>
  <preferencias>
    <fichero>preferencias_agente2.py</fichero>
  </preferencias>
  <oferta>
    <tipo>1</tipo>
  </oferta>
  <futilidad>
    <tipo>1</tipo>
  </futilidad>
  <concesion>
    <tipo>2</tipo>
    <parametros>0.5 1 absoluto</parametros>
    <inicial>1</inicial>
  </concesion>
  <aceptacion>
    <tipo>2</tipo>
    <aceptar>0.3</aceptar>
  </aceptacion>
  <tiempo>300</tiempo>
</agente>
```