8 UTILIDADES

Hemos revisado los conceptos de interacción y comunicación en las sesiones anteriores, así que estamos en posición de definir el concepto de **SMA** como un conjunto de agentes situados en el mismo medio ambiente, los cuales interaccionan entre si comunicándose, y cuyas esferas de influencia, aunque pueden coincidir, son generalmente diferentes. Por **esfera de influencia** entendemos que los agentes pueden controlar o al menos influenciar una parte del medio ambiente. Los agentes de un SMA están sujetos a otras relaciones, como las institucionales, normativas, etc., como se ha ilustrado en la Figura 8.1. Interacciones y encuentros pueden abordarse desde una perspectiva **económica**, con base en el concepto de utilidad. Para ello es necesario introducir un poco de notación.

Sistema Multi-Agentes

Esfera de influencia

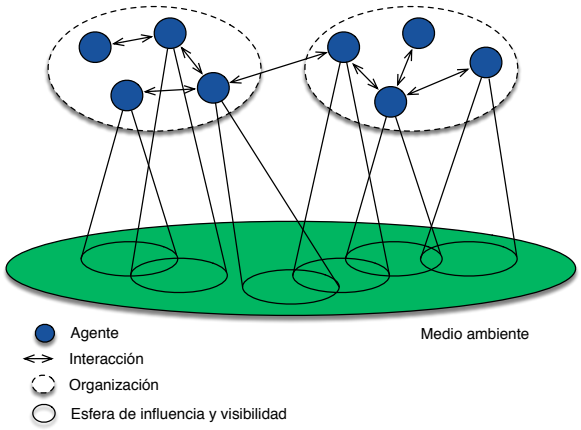

Figura 8.1: Esferas de influencia en un SMA.

8.1 UTILIDADES Y PREFERENCIAS

Por simplicidad, asumiremos que tenemos sólo dos agentes en nuestro SMA: $Ag = \{i, j\}$. Asumamos también que nuestros agentes tienen sus propios intereses, es decir, que tienen **preferencias** y deseos sobre como debería ser el medio ambiente. Por el momento no consideraremos de donde vienen esas preferencias, sólo asumiremos que existen. El conjunto $\Omega = \{w_1, w_2, \dots\}$ representa el posible resultado de las acciones de estos agentes, o estados del medio ambiente que resultan de ello. Por ejemplo piensen que Ω incluye los posibles resultado de un juego donde participan los dos agentes. De esta manera, las preferencias de los agentes pueden capturarse en las **funciones de utilidad** siguientes:

Preferencias

Función de utilidad

$$u_i : \Omega \to \Re$$
$$u_j : \Omega \to \Re$$

Las funciones de utilidad inducen **ordenes de preferencia** sobre las salidas de los agentes. Si ω y ω' son dos posibles salidas en Ω y $u_i(\omega) \geq u_i(\omega')$, se dice que ω es **preferida** por el agente i al menos tanto como ω'. Podemos introducir notación para expresar estos ordenes de preferencia: $\omega \succeq_i \omega'$ significa que $u_i(\omega) \geq_i u_i(\omega')$; y $\omega \succ_i \omega'$ significa que $u_i(\omega) > u_i(\omega')$. *Orden de preferencia*

Observen que la relación de **preferencia** ($\succeq_i$) es un orden sobre Ω puesto que es: *Preferencia*

- **Reflexiva.** Para todo $\omega \in \Omega$ tenemos que $\omega \succeq_i \omega$.
- **Transitiva.** Si $\omega \succeq_i \omega'$ y $\omega' \succeq_i \omega''$ entonces $\omega \succeq_i \omega''$.
- **Comparable.** Para todo $\omega \in \Omega$ y $\omega' \in \Omega$ tenemos que $\omega \succeq_i \omega'$ o bien $\omega' \succeq_i \omega$.

En cambio la relación de **preferencia estricta** ($\succ_i$) satisface las dos últimas propiedades, pero evidentemente no es reflexiva. *Preferencia estricta*

8.1.1 Dinero y utilidad

Sin lugar a dudas, la manera más común de pensar sobre el concepto de utilidad suele estar relacionada con el dinero: entre más dinero, mejor. Sin embargo, la utilidad no es dinero, aunque éste puede funcionar como una analogía útil. La relación típica entre utilidad y dinero se muestra en la Figura 8.2.

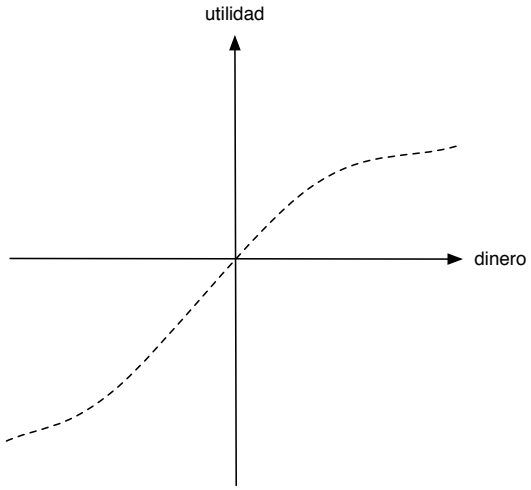

Figura 8.2: La relación entre la utilidad y el dinero

Para entender la gráfica consideren el siguiente ejemplo. Supongamos que tengo una cuenta en el banco de 500 millones de pesos, mientras que tu no tienes ni un centavo. Un benefactor decide que donará generosamente a alguno de nosotros un millón de pesos. Si el benefactor decide donarme el dinero ¿En cuanto se incrementó mi utilidad? Si bien existe un incremento, puesto que ahora tengo 501 millones en el banco, hay pocas cosas que puedo hacer con 501 millones y no las puedo hacer con 500: El incremento ha sido marginal. En el cambio, si el benefactor te da a ti el dinero, hay una diferencia

enorme entre las cosas que se pueden hacer con un millón de pesos en el banco y sin un centavo: El incremento de utilidad ha sido importante. Lo mismo aplica si tengo una deuda de 500 millones, tu no tienes deuda y nuestro benefactor repite su oferta.

8.2 ENCUENTROS

Los agentes en un SMA pueden elegir simultáneamente que acción llevar a cabo y como resultado de estas acciones obtendremos una de las salidas en Ω. De tal forma que la salida actual depende de la combinación de las acciones de los agentes, lo cual se conoce como un **encuentro**. Asumamos que los agentes tiene un repertorio de dos acciones: cooperar (C) y traicionar (D) (*defect*). La dinámica del ambiente se obtiene mediante una **función de transición entre estados**:

Encuentro

Transición entre estados

$$\tau : Ac_i \times Ac_j \to \Omega$$

donde $Ac_i \in \{C, D\}$ es la acción del agente i y $Ac_j \in \{C, D\}$ es la acción del agente j. Por ejemplo:

$$\tau(D, D) = \omega_1,\ \tau(D, C) = \omega_2,\ \tau(C, D) = \omega_3,\ \tau(C, C) = \omega_4$$

donde el ambiente asocia una **salida diferente** a cada combinación de acciones de los agentes i y j. Este ambiente es por lo tanto, **sensible a las acciones** de cada agente. En el otro extremo podemos considerar un ambiente que asocia el **mismo estado** a todas las combinaciones de acciones posibles:

$$\tau(D, D) = \omega_1,\ \tau(D, C) = \omega_1,\ \tau(C, D) = \omega_1,\ \tau(C, C) = \omega_1$$

donde **ninguno** de los agentes tiene influencia sobre el medio ambiente. También es posible considerar el caso donde el ambiente es sensible a las acciones de **uno** de los agentes:

$$\tau(D, D) = \omega_1,\ \tau(D, C) = \omega_2,\ \tau(C, D) = \omega_1,\ \tau(C, C) = \omega_2$$

donde el ambiente es controlado por por el agente j. Si el agente j decide traicionar (D), la salida será ω_1 y si decide cooperar (C) será ω_2.

La historia se vuelve interesante cuando consideramos el ambiente junto con las preferencias de los agentes. Consideren el caso donde ambos agentes pueden influenciar el medio ambiente. Ahora supongamos que los agentes tienen las funciones de utilidad siguientes:

$$\begin{array}{llll} u_i(\omega_1) = 1 & u_i(\omega_2) = 1 & u_i(\omega_3) = 4 & u_i(\omega_4) = 4 \\ u_j(\omega_1) = 1 & u_j(\omega_2) = 4 & u_j(\omega_3) = 1 & u_j(\omega_4) = 4 \end{array}$$

Puesto que sabemos que cada combinación diferente de elecciones de los agentes produce un resultado diferente, podemos abusar un poco de la notación, de forma que:

$$\begin{array}{llll} u_i(D, D) = 1 & u_i(D, C) = 1 & u_i(C, D) = 4 & u_i(C, C) = 4 \\ u_j(D, D) = 1 & u_j(D, C) = 4 & u_j(C, D) = 1 & u_j(C, C) = 4 \end{array}$$

Ahora podemos caracterizar las preferencias del agente i sobre el posible resultado de sus acciones:

$$C, C \succeq_i C, D \succ_i D, C \succeq_i D, D$$

Ahora consideremos la siguiente pregunta: **¿Si fueras el agente i que elegirías: cooperar o traicionar?** En este caso, la respuesta no tiene ambigüedad, el agente prefiere las situaciones donde coopera a aquellas donde traiciona. La opción del agente i es cooperar independientemente de lo que haga el agente j. De igual manera, el agente j prefiere las situaciones donde coopera. Observen que en este caso, ninguno de los agentes debe preocuparse por lo que hace el otro, la acción que eligen no depende de la acción del otro. Si ambos agentes actúan racionalmente, ejecutarán la acción que los lleva a las situaciones preferidas, por lo que la acción conjunta será C, C: ambos agente cooperarán.

Ahora supongamos que, para el mismo medio ambiente, las funciones de utilidad son las siguientes:

$$\begin{array}{llll} u_i(D,D) = 4 & u_i(D,C) = 4 & u_i(C,D) = 1 & u_i(C,C) = 1 \\ u_j(D,D) = 4 & u_j(D,C) = 1 & u_j(C,D) = 4 & u_j(C,C) = 1 \end{array}$$

Las preferencias del agente i son como sigue:

$$D, D \succeq_i D, C \succ_i C, D \succeq_i C, C$$

En este escenario, el agente i no puede hacer nada mejor que traicionar. El agente prefiere las salidas donde traiciona a todas aquellas donde coopera. De igual manera, el agente j no puede hacer nada mejor que traicionar. Nuevamente los agentes no tienen que dedicar recursos a elaborar una estrategia basada en qué hará el otro agente. Aunque claro, en la mayoría de los SMA los encuentros no son tan claros como en estos dos ejemplos.

Las situaciones anteriores pueden resumirse en una notación estándar utilizada en la Teoría de Juegos [187, 159, 5], las **matrices de pago**: *Matriz de pago*

	D	C
D	4,4	4,1
C	1,4,	1,1

La manera de leer una matriz de pago es como sigue. Cada una de las cuatro celdas en la matriz corresponde a una de las cuatro posibles salidas. Por ejemplo, la celda de arriba a la derecha, corresponde a la salida donde i coopera y j traiciona; la celda de abajo a la izquierda corresponde a la situación donde i traiciona y j coopera. Los pagos recibidos por los agentes se escriben en cada celda. El primer valor de cada celda corresponde al pago recibido por el agente j (el jugador del renglón); mientras que el segundo valor corresponde al pago obtenido por el agente i (el jugador en la columna). Esta es la notación que utilizaremos en el resto de la sesión.

8.3 ESTRATEGIAS DOMINANTES Y EQUILIBRIO DE NASH

Dado un encuentro SMA entre dos agentes i y j, existe una pregunta crítica que ambos quieren contestar: ¿Qué debo hacer? Hemos presentado algunos encuentros, e informalmente hemos argumentado sobre cual es la mejor opción posible. En esta sección definiremos algunos conceptos necesarios para responder esta cuestión formalmente.

Dominación

El primer concepto es el de **dominación**. Supongamos que tenemos dos subconjuntos de Ω a los que nos referiremos como Ω_1 y Ω_2 respectivamente. Diremos que Ω_1 **domina** a Ω_2 para el agente i si cada estado de Ω_1 es preferido por i sobre cualquier estado de Ω_2. Por ejemplo, supongamos que:

- $\Omega = \{\omega_1, \omega_2, \omega_3, \omega_4\}$
- $\omega_1 \succ_i \omega_2 \succ_i \omega_3 \succ_i \omega_4$
- $\Omega_1 = \{\omega_1, \omega_2\}$
- $\Omega_2 = \{\omega_3, \omega_4\}$

Es evidente que Ω_1 domina fuertemente a Ω_2. Formalmente, un conjunto de estados Ω_1 **domina fuertemente** a un conjunto Ω_2 si la siguiente condición es verdadera, para el agente i:

$$\forall \omega_1 \in \Omega_1, \forall \omega_2 \in \Omega_2 \; \omega_1 \succ_i \omega_2$$

Estrategia

Para homogeneizar nuestro vocabulario con el de la Teoría de Juegos, nos referiremos a las acciones de los agentes (miembros de Ac) como **estrategias**. Dada una estrategia particular s para un agente i en un escenario de interacción SMA, habrá un número posible de estados resultantes. Denotaremos s^* a los estados que resulten de la estrategia s del agente i. En los ejemplos de la sección anterior, desde el punto de vista del agente i tenemos que $C^* = \{\omega_3, \omega_4\}$ y $D^* = \{\omega_1, \omega_2\}$.

Diremos que la estrategia s_1 domina a la estrategia s_2 si el conjunto de estados resultantes al jugar con la estrategia s_1 domina al conjunto de estados resultantes al jugar con la estrategia s_2. Esto es si s_1^* domina a s_2^*. La presencia de una **estrategia dominante**, hace la elección de qué hacer fácil. El problema es que normalmente hay más de una estrategia dominante y es necesario usar la relación de dominio débil.

Estrategia dominante

Equilibrio de Nash

El otro concepto relevante es la noción de equilibrio, o para ser más específicos, el **equilibrio de Nash**. Decimos que dos estrategias están en equilibrio si:

1. Bajo el supuesto de que el agente i jugará con la estrategia s_1, el agente j no puede hacer nada mejor que jugar con la estrategia s_2; y
2. Bajo el supuesto de que el agente j jugará con la estrategia s_2, el agente i no puede hacer nada mejor que jugar con la estrategia s_1.

Limitantes

La forma mutua del equilibrio es importante porque fija a los agentes en un par de estrategias. Ningún agente tiene incentivo alguno para salir del equilibrio de Nash. Desafortunadamente hay dos **limitantes** de la teoría de juegos con respecto al equilibrio:

1. No todo escenario de interacción tiene un equilibrio de Nash; y
2. Algunos escenarios de interacción tienen más de un equilibrio de Nash.

A pesar de estos resultados negativos, el equilibrio de Nash es un concepto extremadamente importante en el análisis de los SMA.

8.4 INTERACCIONES COMPETITIVAS Y DE SUMA CERO

Supongamos que tenemos un escenario donde el estado $\omega \in \Omega$ es preferido por el agente i con respecto a un estado ω', si y sólo si ω' es preferido por el agente j con respecto a ω. Formalmente:

$$\omega \succ_i \omega' \text{ Si y sólo si } \omega' \succ_j \omega$$

Interacción estrictamente competitiva

Una interacción **estrictamente competitiva** es aquella donde, como en el caso anterior, las preferencias de los agentes son entonces diametralmente opuestas, de forma que un agente puede incrementar su utilidad sólo a costa del otro agente.

Encuentro de suma cero

Los encuentros de **suma cero** son aquellos en los cuales para cualquier estado alcanzado, las utilidades de los dos agentes suman cero. Formalmente:

$$\forall \omega \in \Omega \quad u_i(\omega) + u_j(\omega) = 0$$

Es evidente que todo encuentro de suma cero es estrictamente competitivo. Estos escenarios son interesantes porque son el caso más viciado de interacción, donde no hay posibilidad de comportamiento cooperativo. Si un agente permite que el otro agente tenga utilidad positiva, el tendrá utilidad negativa y estará peor que antes de llevar a cabo la interacción.

Los juegos como las damas y el ajedrez son los ejemplos más claros de este tipo de interacción. Es difícil encontrar ejemplos de suma cero en el mundo real. Inclusive en las guerras, algunos acuerdos pueden encontrarse (por ejemplo, no destruir el planeta). A pesar de ello, en muchas interacciones nos comportamos como si fuesen de suma cero.

8.5 EL DILEMA DEL PRISIONERO

Consideren el siguiente escenario: Dos hombres son acusados de un crimen colectivo y se les mantiene en celdas separadas. No tienen forma de comunicarse entre si o de llegar a alguna forma de acuerdo. A ambos se les dice que:

1. Si uno de ellos confiesa el crimen y el otro no lo hace, el confeso será liberado y al otro se le darán tres años de cárcel.
2. Si ambos confiesan el crimen estarán dos años en la cárcel.

Ambos prisioneros saben que si ninguno confiesa el crimen, entonces cada uno será encarcelado un año. Nos referiremos a la confesión como traición

(D) y a la no confesión como cooperación (C). Antes de continuar, reflexionen sobre qué harían ustedes.

Hay cuatro salidas posibles al dilema del prisionero, dependiendo de si los agentes cooperan o traicionan. La matriz de pago correspondiente se muestra a continuación:

	D	C
D	2,2	0,5
C	5,0	3,3

Observen que los números en la matriz no se refieren a los años que los agentes pasarían en prisión, sino a que tan bueno el resultado es para los agentes. El menor tiempo en la cárcel, lo mejor para le agente. En otras palabras las utilidades son:

$$u_i(D,D) = 2 \quad u_i(D,C) = 5 \quad u_i(C,D) = 0 \quad u_i(C,C) = 3$$
$$u_j(D,D) = 2 \quad u_j(D,C) = 0 \quad u_j(C,D) = 5 \quad u_j(C,C) = 3$$

y sus preferencias son:

$$D,C \succ_i C,C \succ_i D,D \succ_i C,D,$$
$$C,D \succ_j C,C \succ_j D,D \succ_j D,C.$$

¿Qué deberían hacer los prisioneros? En este caso no es cierto que un agente prefiera cooperar por sobre todas sus demás opciones, pero tampoco prefiere traicionar sobre todas las demás opciones. El enfoque estándar es ponerse en el lugar de uno de los agentes, digamos i y razonar de la siguiente manera:

- Supongamos que i coopera. Entonces si j coopera, ambos agentes obtendrán un pago igual a tres. Pero si j traicionara, el agente i obtendría un pago de cero, de forma que el mejor pago que i puede garantizar cooperando es cero.
- Supongamos que i traiciona. Entonces si j coopera, i obtendrá un pago igual a cinco, mientras que si j traicionará, obtendría un pago igual a dos. De forma que el mejor pago que puedo garantizar es dos.
- De forma que si i coopera, en el peor de los casos obtendrá un pago de cero; mientras que si traiciona, en el peor de los casos obtendrá un pago igual a dos.
- El agente i preferirá garantizar un pago igual a dos, que obtener un pago igual a cero y por lo tanto, traicionará al agente j.

Puesto que el escenario es **simétrico**, i.e., ambos agentes razonan de la misma manera, entonces el estado obtenido si ambos agentes actúan racionalmente es que ambos agentes traicionan, obteniendo así un pago igual a dos.

Aunque la intuición nos diga que lo mejor que pueden hacer ambos agentes es cooperar, tenemos que si i asume que j cooperará, lo mejor que pude hacer es traicionar. El equilibrio de Nash en este escenario es que ambos agentes traicionen, aunque esto los lleve a **perder** utilidad, de ahí que sea un dilema. *Dilema*

8.5.1 La sombra del futuro

Consideremos ahora la idea de jugar repetidamente el dilema del prisionero. En el **dilema del prisionero iterado**, el juego es llevado a cabo un cierto número de rondas. Se asume que cada agente puede ver lo que hizo el otro agente en la ronda anterior: el agente i puede ver si j traicionó o no; y el agente j puede ver si i traicionó o no.

Dilema del prisionero iterado

Ahora consideremos que el juego se repite indefinidamente, ronda tras ronda. Bajo este supuesto, ¿Cual es la decisión racional? Si i sabe que enfrentará al mismo oponente en todos los juegos, sus razones para traicionar se desvanecen por las siguiente dos razones:

- Si i traiciona ahora, el oponente j puede castigarle traicionando más tarde. El **castigo** no es posible en la definición no iterada del dilema.
- Si i tantea cooperar y recibe el pago del tonto en el juego en la primera ronda, entonces debido a que se juega indefinidamente, esta perdida de utilidad puede ser amortizada en los juegos futuros. En la perspectiva de que el juego es infinito, la perdida de una sola unidad de utilidad representa un pequeño porcentaje de la utilidad total ganada.

Castigo

En la versión iterada del dilema del prisionero, la sombra del futuro hace que los agentes cooperen. Esto es una excelente nueva, pues en la vida real, es muy probable que si interaccionamos con alguien en repetidas ocasiones, la cooperación racional sea posible. Sin embargo hay un problema: Supongamos que aceptamos jugar la versión iterada del dilema del prisionero un número **finito** de veces, digamos 100. Es necesario saber de antemano, presumiblemente, la estrategia que utilizaremos al jugar. Consideremos ahora la última ronda (la 100). Ahora, en esa ronda, tu sabrás, al igual que el oponente, que no interaccionarás más. En otras palabras, esa ronda se comporta como la versión no iterada del dilema del prisionero. Como hemos visto, ambos jugadores traicionarán. Por lo tanto, la última ronda real es la 99, pero el mismo análisis se puede aplicar y por inducción los agentes traicionarán cuando el dilema del prisionero se juega un número finito de veces y ese número es conocido por ambos agentes.

Iteraciones finitas

Este resultado no es tan malo como podría parecer. Los agentes cooperarán sin necesidad de jugar indefinidamente, a condición de que la sombra del futuro se extienda lo suficiente, esto es, que los agentes crean que existe una posibilidad de volver a interaccionar en el futuro. La otra razón es que aunque un agente cooperativo pueda perder al jugar con un agente que traiciona, su utilidad total puede ser buena si aseguramos que pueda interaccionar con otros agentes cooperativos. Para entender esto veremos al torneo del dilema del prisionero propuesto por Axelrod.

8.5.2 El torneo de Axelrod

Axelrod [9] es un politólogo interesado en como la cooperación puede emerger en una sociedad de agentes individualistas. En 1980 organizó un torneo público en el cual politólogos, psicólogos, economistas y teóricos del juego fueron invitados a enviar un programa que jugará el dilema del prisionero

iterado. Cada programa de computadora tenía disponible las opciones previas de su contrincante y simplemente seleccionada C o D basado en esa información. Cada programa jugaba contra los demás programas cinco juegos de 200 rondas. El ganador era el programa que que lo hiciera mejor sobre todos los demás. Los programas recibidos iban de 152 línea de código a sólo cinco. A continuación se presentan algunas de las **estrategias** usadas: *Estrategias*

- **Todo-D.** Esta estrategia sigue los preceptos teóricos: la estrategia racional es siempre traicionar, independientemente de lo que el otro haga.
- **Aleatoria.** Esta estrategia es un control: ignora que ha hecho el adversario en la ronda anterior y selecciona C o D aleatoriamente, con la misma probabilidad. Una buena estrategia debería tener un desempeño mejor a la aleatoria.
- **Ojo por ojo (tit for tat).** Esta estrategia es como sigue:
  1. En la primera ronda el agente coopera;
  2. en las rondas $t > 1$ hace lo que su oponente hizo en la ronda $t - 1$.

  Su implementación requirió solo cinco línea de código Fortran.
- **Probador.** Esta estrategia intentaba explotar a aquellos programas que no castigaban la traición. Como su nombre lo sugiere, en la primer ronda evaluaba a su oponente traicionándolo. Si el oponente contestaba traicionado, en las rondas subsecuentes el agente jugaba ojo por ojo. Si el otro agente no traicionaba, continuaba jugando dos rondas cooperando y luego traicionando una vez nuevamente.
- **Joss.** Como el probador, la estrategia de Joss (ídolo chino) intenta explotar la debilidad de sus oponentes. Básicamente se comporta siguiendo una estrategia ojo por ojo, sólo que traiciona el 10 % del tiempo, en lugar de cooperar.

Antes de continuar, consideren las siguientes dos preguntas:

1. ¿Qué estrategia consideras es la mejor?
2. Si tu participarás en el concurso, ¿Qué estrategia aplicarías?

La estrategia **ganadora** fue la más simple: Ojo por ojo. Aunque esto parece ser una prueba empírica de que el análisis teórico del dilema del prisionero iterado está equivocado, el resultado es más sutil. La estrategia ojo por ojo gana porque se evalúa el escore sobre todas las estrategias contra las cuales juega. Cuando ojo por ojo juega contra todo-D: el resultado es el esperado, todo-D gana. Ojo por ojo no es la estrategia óptima para el dilema del prisionero iterado, su éxito radica en que tiene oportunidad de jugar con otros agentes que también cooperan. Axelrod intenta explicar las razones del éxito de ojo por ojo, como sigue: *Estrategia ganadora*

1. No seas envidioso. En el dilema del prisionero no es necesario derrotar al oponente para jugar bien.

2. No seas el primero en traicionar. Aunque comenzar a jugar cooperando es riesgoso, la pérdida de traición en la primera ronda es pequeña comparada con las posibilidades de cooperar.

3. Se reciproco en la cooperación y la traición. La estrategia ojo por ojo representa un balance entre castigar y ser perdonado: la combinación de castigar la traición y premiar la cooperación parece motivar la cooperación.

4. No seas demasiado listo. La estrategia ojo por ojo es la más simple de todas las estrategias y programas presentados. Axelrod explica su éxito de la siguiente forma:

   a) Los programas más complejos intentan crear un modelo del comportamiento del otro agente, ignorando el hecho de que el otro agente está de observando al agente original – carecen de un modelo de aprendizaje recíproco, que es lo que sucede en realidad.

   b) Los programas más complejos sobre generalizan cuando encuentran el defecto del oponente y no permiten que la cooperación sea aún posible en el futuro – no perdonan.

   c) Muchos de los programas complejos exhibían un comportamiento demasiado complicado para ser entendido por el otro agente – el oponente creía que su comportamiento era aleatorio.

8.6 CASO DE ESTUDIO: DILEMA DEL PRISIONERO ITERADO

El proyecto que desarrollaremos está basado en el código para el dilema del prisionero iterado que acompaña la distribución de Jason [1]. Primero, todos los jugadores conocen la matriz de pago; es más se asume que la matriz de pago es conocimiento común. Para ello definimos la matriz como una relación $u/4$ en el archivo `matrizPago.asl`. Este código será importado en el código común a todos los jugadores, independientemente de la estrategia que usen. La matriz de pago se define de la siguiente manera:

```
1 /* Matriz de Pago
2  * formato: u(Acción_ag1,Acción_ag2,Utilidad_ag1,Utilidad_ag2)
3  * Usada por todos los agentes jugadores (Conociminto común)
4 */
5
6 u(c,d,0,5). // si traiciono al que coopera tomo 5
7 u(d,c,5,0). // si me traicionan cooperando me voy con 0
8 u(c,c,3,3). // recompensa por cooperación mutua = 3
9 u(d,d,2,2). // castigo por traición mutua = 2
```

La parte común de los jugadores permite a este tipo de agentes interactuar con un arbitro que hace el papel del medio ambiente en esta implementación. El código común de todos los jugadores se encuentra en el archivo `jugador.asl` y se define como sigue:

1 `/examples/iterated-prisoners-dilemma`

```
/*** creencias iniciales ***/

mi_utilidad(0).

// Matriz de pago conocida por todos los agentes
{include("matrizPago.asl")}

/*** metas iniciales ***/

!inicio.

/*** planes ***/

// Me presento con el arbitro
+!inicio <-
  .my_name(Yo);
  .send(arbitro, tell, jugador(Yo)).

// El resultado del encuentro ha sido anunicado por el
// arbitro: actualizar score y registrar resultados por
// si mi estrategia lo requiere
@u[atomic]
+u(Paso,Utilidad)[source(arbitro)] : arrestado(Paso,Oponente) <-
  -u(_,_);
  +u(Paso,Utilidad);
  ?mi_utilidad(U);
  UtilidadNueva = U+Utilidad;
  -+mi_utilidad(UtilidadNueva); // actualizar utilidad total
  -arrestado(Paso,Oponente)[source(arbitro)]; // olvidar mi arresto
  -u(Paso,Utilidad)[source(arbitro)]; // olvidar utilidades, no las necesito
  ?u(_,AccOponente,Utilidad,_); // Si obtuve Utilidad el otro actuó AccOponente
  !registrar(Paso,Oponente,AccOponente); // algunas estrategias lo requiren
  dpi.plot(Paso,UtilidadNueva);  // acción interna para graficar utilidades
  .print("Obtuve ",Utilidad," en el paso ",Paso,". Mi total es ahora: ",
         UtilidadNueva).
```

Todo jugador tiene una utilidad inicial de cero que se modificará con los encuentros que tenga. La matriz de pago se incluye para todos los agentes. La meta inicial de un jugador es presentarse ante el agente `arbitro` para que éste le pueda asignar juegos. ¿Qué opción elige cada agente? Eso depende de su estrategia que definirá el código completo de cada agente jugador. Independientemente de su elección, una vez celebrado un encuentro, el `arbitro` computará la utilidad obtenida por cada agente y se las comunicará mediante un acto de habla. En respuesta a este informe del `arbitro`, el jugador actualizará su utilidad y el registro de encuentros pasados, si esto es necesario. También utilizará la acción interna `dpi.plot` para graficar su utilidad.

El código del arbitro es como sigue:

```
/* creencias iniciales */

paso(0).
rondas(300).

// Creencias comunes sobre la utilidad del dilema del prisionero
{include("matrizPago.asl")}

// Regla para seleccionar dos jugadores al azar
```

```
seleccionar(Jugador1,Jugador2) :-
  .count(jugador(_),N)  &
  dos_rands(R1,R2,N) &
  .findall(J,jugador(J),ListaJugadores) &
  .nth(R1,ListaJugadores,Jugador1) & .nth(R2,ListaJugadores,Jugador2).

// Regla para obtener dos números aleatorios "diferentes" menores a N
// dpi.random es una acción interna con backtracking
dos_rands(R1,R2,N)   :- dpi.random(R1,N) & dpi.random(R2,N) & R1 \== R2.

/* planes */

// Tan pronto como sepa de 2 jugadores puedo arrestarlos
+jugador(_) :  .count(jugador(_),NoAgs) & NoAgs >= 2 <-
  !!arrestar.

// Tengo la meta de arrestar a dos jugadores
@arrestar[atomic]
+!arrestar : paso(Paso) & rondas(Ronda) & Paso <= Ronda <-
  ?seleccionar(Jugador1,Jugador2); // Selecciona dos jugadores al azar
  .print("Arrestando a ",Jugador1," y ",Jugador2," (paso ",Paso,")");
  .send(Jugador1,tell,arrestado(Paso,Jugador2));
  .send(Jugador2,tell,arrestado(Paso,Jugador1));
  -+paso(Paso+1).

// He terminado el número de rondas establecidas: no hago nada.
+!arrestar.

// Qué jugó el segundo de los jugadores?
// De forma que pueda calcular las utilidades y comunicarlas
@jugar[atomic]
+jugar(Paso,AccAg1)[source(Jugador1)]
  :  jugar(Paso,AccAg2)[source(Jugador2)] & Jugador1 \== Jugador2 <-
  ?u(AccAg1,AccAg2,UAg1,UAg2);
  .print("Utilidades en el paso ",Paso,": ",UAg1," ",UAg2);
  .send(Jugador1,tell,u(Paso,UAg1));
  .send(Jugador2,tell,u(Paso,UAg2));
  .abolish(jugar(Paso,_)[source(_)]);
  !!arrestar.

+jugar(Paso,_). // si solo conozco una jugada no hago nada.
```

Las creencias iniciales del `arbitro` se usan para controlar el ciclo de control: la creencia *rondas*/1 especifica el número de rondas a jugar, mientras que *paso*/1 es un contador. Como el resto de los agentes, el arbitro cree la matriz de pago que todos conocen. La regla *seleccionar*/2 escoge dos jugadores dentro del conjunto jugadores conocidos, para celebrar un encuentro entre ellos. La acción interna `dpi.random` es un random con backtracking.

El `arbitro` está a la espera de que dos jugadores sean conocidos, en ese momento procede a realizar arrestos. Este proceso consiste en seleccionar dos jugadores aleatoriamente, avisarles de que han sido arrestados e incrementar el contador en un paso. El aviso de arresto incluye como argumentos: el paso en que ha sucedido el arresto y el nombre del agente complice (el oponente, si la situación se ve como un juego).

Cuando el primer jugador envía su acción elegida, el `arbitro` procede a computar las utilidades obtenidas por los dos agentes y a comunicárselas. Revisemos ahora las estrategias.

Todo D

Este agente traiciona siempre. Su código es como sigue:

```
/*** Soy un jugador.. ***/

{ include("jugador.asl") }

/*** Y mi estrategia es ... ***/

// Siempre traicionar
+arrestado(Paso,Oponente)[source(arbitro)] <-
  .send(arbitro, tell, jugar(Paso,d));
  .print("Traiciono como siempre, a ",Oponente," en el paso ",Paso).

// No necesito registros: no hace nada
+!registrar(Paso,Oponente,AccOponente).
```

8.6.1 TodoC

Este agente coopera siempre. Su código es como sigue:

```
/*** Soy un jugador...  ***/

{ include("jugador.asl") }

/*** Y mi estrategia es ... ***/

// Siempre cooperar
+arrestado(Paso,Oponente)[source(arbitro)] <-
  .send(arbitro, tell, jugar(Paso,c));
  .print("Coopero como simpre con ",Oponente," en el paso ",Paso).

// No necesito registros nuevoss
+!registrar(Paso,Oponente,AccOponente).
```

8.6.2 Ojo por Ojo (*Tit-forTat*)

Este agente es cooperativo la primera vez que juega y lleva un registro de la última jugada de su oponente. El juega de la misma manera. Su código es:

```
/*** Soy un jugador... ***/

{include("jugador.asl")}

/*** Que necesita acordarse del juego del oponente ... ***/

@registrar[atomic]
+!registrar(Paso,Oponente,AccOponente) <-
  -ultimo_mov(Oponente,_); // NB: cannot use -+ here!
  +ultimo_mov(Oponente,AccOponente).

/*** Mi estrategia es... ***/

// Ojo por ojo y diente por diente
// (1) Juego lo mismo que  jugo el oponente la última vez
+arrestado(Paso,Oponente)[source(arbitro)]
```

```
17    :  ultimo_mov(Oponente,UltimaJugadaOponente)
18    <-
19    .send(arbitro, tell, jugar(Paso,UltimaJugadaOponente));
20    .print("Jugué ",UltimaJugadaOponente," contra ",Oponente,
21           " en el paso ",Paso).
22
23  // (2) Soy cortés la primera vez que juego (coopero)
24  +arrestado(Paso,Oponente)[source(arbitro)] <-
25    .send(arbitro, tell, jugar(Paso,c));
26    .print("Cooperé contra ",Oponente," en el paso ",Paso).
```

8.6.3 Resultados

Con estos agentes es posible recuperar los resultados de Axelrod en una simulación. Si ponemos a jugar a un agente de cada tipo, es claro que traicionar paga, como puede verse en la Figura 8.3.

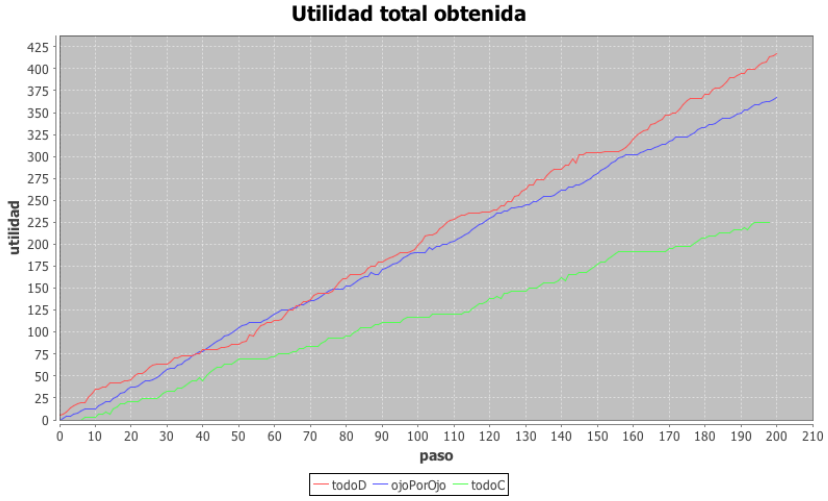

Figura 8.3: Las utilidades obtenidas en un juego con un agente de cada tipo. Siempre traicionar (todoD) gana.

Esto se debe, como sabemos, a que el agente que siempre traiciona sobre explota al agente que siempre coopera, sacando ventaja sobre la estrategia de Ojo por Ojo, que es más equilibrada. La conclusión es que hay que ser justo en las decisiones y eso implica no aceptar cooperar con alguien que te traiciona. En la siguiente simulación, he eliminado al agente que siempre coopera e incluído tres agentes Ojo por Ojo y tres que siempre traicionan. El resultado es que los agentes tienen más probabilidad de tener encuentros cooperativos, que pagan mejor, y la estrategia Ojo po Ojo puede resultar la ganadora, como se muestra en la figura 8.4.

Código suplementario

Los agentes hacen uso de dos acciones definidas por el usuario. Con la primera, `dpi.random`, se pueden generar números aleatorios con backtracking. Su código es como sigue:

```
1  package dpi;
2
```

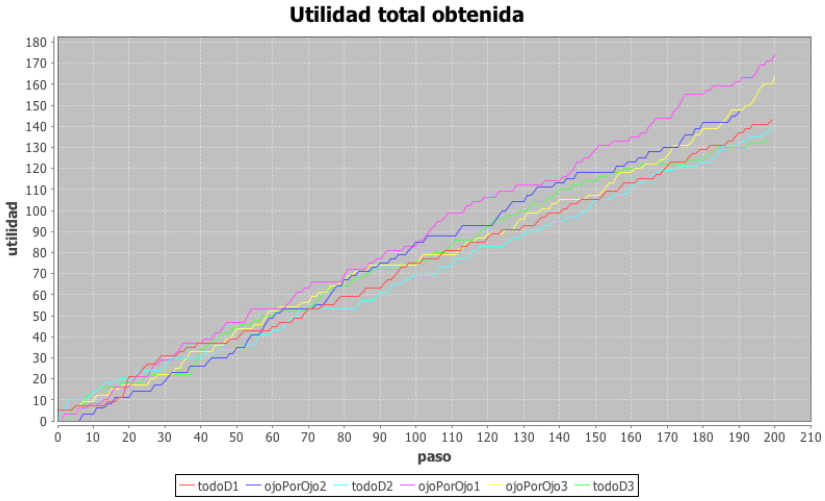

Figura 8.4: Las utilidades obtenidas en un juego sin el agente que siempre coopera y tres agentes `todoD` y `ojoPorOjo`. En este caso Ojo por Ojo gana, aunque esto no sucede siempre.

```
3  import jason.JasonException;
4  import jason.asSemantics.DefaultInternalAction;
5  import jason.asSemantics.TransitionSystem;
6  import jason.asSemantics.Unifier;
7  import jason.asSyntax.NumberTermImpl;
8  import jason.asSyntax.NumberTerm;
9  import jason.asSyntax.Term;
10
11 import java.util.Random;
12 import java.util.Iterator;
13
14 /** version de random con backtrack */
15 public class random extends DefaultInternalAction {
16
17   private static final long serialVersionUID = 2L;
18   private Random random = new Random();
19
20   @Override
21   public Object execute(final TransitionSystem ts, final Unifier un,
22       final Term[] args) throws Exception {
23     try {
24       if (!args[0].isVar()) {
25         throw new JasonException("El primer argumento de la acción interna "
26             + "'random' no es una variable.");
27       }
28       if (!args[1].isNumeric()) {
29         throw new JasonException("El segundo argumento de la acción interna "
30             + "'random' no es un número.");
31       }
32       final int max = (int) ((NumberTerm) args[1]).solve();
33
34       return new Iterator<Unifier>() {
35
36         // Siempre tenemos un número aleatorio siguiente
37         public boolean hasNext() {
38           return ts.getUserAgArch().isRunning();
39         }
40
```

```
41          public Unifier next() {
42            Unifier c = un.clone();
43            c.unifies(args[0], new NumberTermImpl(random.nextInt(max)));
44            return c;
45          }
46
47          public void remove() {
48          }
49        };
50
51      } catch (ArrayIndexOutOfBoundsException e) {
52        throw new JasonException("La acción interna 'random' recibió un argumento
              ↪ "
53            + "inadecuado.");
54      } catch (Exception e) {
55        throw new JasonException("Error en la acción interna 'random': " + e, e);
56      }
57    }
58  }
```

La segunda permite generar la gráfica de utilidades que presentamos en la sección de resulados. Se llama `dpi.plot` y su código es el siguiente:

```
1  package dpi;
2
3  import jason.asSemantics.DefaultInternalAction;
4  import jason.asSemantics.TransitionSystem;
5  import jason.asSemantics.Unifier;
6  import jason.asSyntax.NumberTerm;
7  import jason.asSyntax.Term;
8
9  import java.util.HashMap;
10 import java.util.Map;
11
12 import javax.swing.JFrame;
13 import javax.swing.JPanel;
14
15 import org.jfree.chart.ChartFactory;
16 import org.jfree.chart.ChartPanel;
17 import org.jfree.chart.JFreeChart;
18 import org.jfree.chart.plot.PlotOrientation;
19 import org.jfree.data.xy.DefaultXYDataset;
20
21 /** Gráfica con la ganancia de todos los jugadores */
22 public class plot extends DefaultInternalAction {
23
24   private static final long serialVersionUID = 2L;
25
26   Map<Integer, Integer> values = new HashMap<Integer, Integer>();
27
28   static DefaultXYDataset dataset = new DefaultXYDataset();
29   static {
30     JFreeChart xyc = ChartFactory.createXYLineChart("Utilidad total obtenida",
31         "paso",
32         "utilidad",
33         dataset, // dataset,
34         PlotOrientation.VERTICAL, // orientación,
35         true, // leyenda,
36         true, // tooltips,
37         true); // urls
38
```

```
39      JPanel panel = new ChartPanel(xyc,false);
40      JFrame frame = new JFrame();
41      frame.setTitle("Dilema del Prisionero Iterado");
42      frame.setSize(800,500);
43      frame.add(panel);
44      frame.pack();
45      frame.setVisible(true);
46    }
47
48    @Override
49    public Object execute(final TransitionSystem ts, final Unifier un,
50        final Term[] args) throws Exception {
51      int step = (int) ((NumberTerm) args[0]).solve();
52      int score = (int) ((NumberTerm) args[1]).solve();
53      addValue(ts.getUserAgArch().getAgName(), step, score);
54      return true;
55    }
56
57    void addValue(String agName, int step, int vl) {
58      values.put(step, vl);
59      double[][] data = getData(step);
60      synchronized (dataset) {
61        dataset.addSeries(agName, data);
62      }
63    }
64
65    private double[][] getData(int maxStep) {
66      double[][] r = new double[2][maxStep + 1];
67      int vl = 0;
68      for (int step = 0; step <= maxStep; step++) {
69        if (values.containsKey(step))
70          vl = values.get(step);
71        r[0][step] = step;
72        r[1][step] = vl;
73      }
74      return r;
75    }
76  }
```

8.7 LECTURAS Y EJERCICIOS SUGERIDOS

Este capítulo es una introducción somera al tema de las preferencias y su uso en los SMA. El material presentado está basado principalmente en trabajo de Wooldridge [192]. Shoham y Leyton-Brown [174] ofrecen una serie de resultados teóricos más profundos y resulta un complemento ideal para estos temas. Apt y Grädel [5] también ofrecen una serie de artículos con más profundidad, orientados a un público proveniente de las Ciencias de la Computación.

Ejercicios

Ejercicio 8.1. *Revisar el artículo de Axelrod [8] y reproducir el experimento del dilema del prisionero iterado con exactitud. Se pueden implementar otras estrategias*

descritas en este artículo. Es necesario implementar una estrategia aleatoria como estrategia control.