Enfoque Computacional para Juegos de Suma Cero

A continuación exploraremos el **enfoque computacional** para **juegos de suma cero** desde la óptica **min-max** o, equivalentemente, **max-min**. Nos enfocaremos especialmente en:

1. **Formulación del problema** en juegos de suma cero.
2. **Idea intuitiva** del *algoritmo minimax* (y por qué es relevante).
3. **Método de resolución** y **pseudocódigo** (principalmente a través de un enfoque de **Programación Lineal**).
4. **Complejidad computacional** (tanto para casos particulares como de forma general).
5. **Ejemplo** concreto de cómo se aplica este método.

---

1. Fundamentos: Juegos de Suma Cero y Minimax

1.1. Juego de Suma Cero

- Un **juego de suma cero** (en forma normal) con dos jugadores (llamados a menudo *Jugador Fila* y *Jugador Columna*) se describe por una **matriz de pagos** A de tamaño $m \times n$.
- Cuando el *Jugador Fila* (1) elige una fila i y el *Jugador Columna* (2) elige una columna j, el pago de Jugador 1 es A_{ij}.
- El pago de Jugador 2 es $-A_{ij}$. Así, la suma de ambos pagos es cero.
- Buscamos **estrategias mixtas** para cada jugador (distribuciones de probabilidad sobre sus acciones puras) que alcancen un **equilibrio de Nash**. Para juegos de suma cero, esto coincide con la solución **minimax** (o **maximin**).

1.2. Teorema Minimax

Von Neumann demostró que en los juegos de suma cero, el **valor de maximin** coincide con el **valor de minimax**, y la estrategia que lo logra es el **equilibrio de Nash**.

- Sea $\mathbf{x}$ la distribución de probabilidad del Jugador Fila sobre sus m filas ($\mathbf{x} \ge 0$, $\sum_i x_i=1$).
- Sea $\mathbf{y}$ la distribución de probabilidad del Jugador Columna sobre sus n columnas ($\mathbf{y}\ge 0$, $\sum_j y_j=1$).
- La utilidad esperada de Jugador 1 (Fila), jugando $\mathbf{x}$ contra $\mathbf{y}$, es $\mathbf{x}^\top A \mathbf{y}$.
- El Jugador 1 quiere **maximizar** este valor; el Jugador 2 quiere **minimizarlo** (equivalente a maximizar $-\mathbf{x}^\top A \mathbf{y}$).

El **Teorema del minimax** indica:
$
\max_{\mathbf{x}\ge 0, \sum x_i=1}
\min_{\mathbf{y}\ge 0, \sum y_j=1} \mathbf{x}^\top A \mathbf{y}
;=;
\min_{\mathbf{y}\ge 0, \sum y_j=1}

\max_{\mathbf{x}\ge 0, \sum x_i=1} \mathbf{x}^\top A \mathbf{y}
;=; v,
$
donde v es el **valor del juego**.

---

2. Idea Intuitiva del Algoritmo Minimax

Motivación:

- El Jugador Fila (1) mezcla sus acciones para **maximizar** su ganancia **asegurada** (independientemente de la estrategia del adversario).
- El Jugador Columna (2) mezcla sus acciones para **minimizar** la ganancia del primero.
- En el equilibrio, ninguno puede mejorar cambiando unilateralmente su mezcla: es la solución que iguala **maximin** y **minimax**.

Idea breve:

- Piense en Jugador 1 intentando "garantizar" que su ganancia sea al menos v.
- Para lograrlo, necesita una estrategia $\mathbf{x}$ tal que **para todas** las posibles columnas $\mathbf{y}$ de Jugador 2, la utilidad sea $\geq v$.
- Jugador 2, por su parte, quiere lograr que la ganancia sea $\leq v$.
- Resolveremos estas condiciones mediante **programación lineal**.

---

3. Formulación y Resolución por Programación Lineal

Existen varias formas de escribir el **problema primal** (para Jugador 1) y su **dual** (para Jugador 2). Aquí damos la formulación típica:

3.1. Modelo en PL (Programación Lineal) para Jugador Fila

Para un **juego de suma cero** con matriz de pagos A (dimensiones $m \times n$):

PROF

Objetivo: Encuentra $\mathbf{x} = (x_1,\dots,x_m)$ y un escalar v tales que:

$
\max_{\mathbf{x},,v} ; v
$
sujeto a las restricciones:

1. $\sum_{i=1}^m x_i = 1$,
2. $x_i \ge 0 \quad \forall i,$
3. $\mathbf{x}^\top A \mathbf{e}_j \ge v \quad$ para cada columna j.
    - ($\mathbf{e}_j$ *es el vector canónico que significa "columna j pura"; en la práctica esto se traduce a* $\sum_i x_i A_{ij} \ge v$).

Interpretación:

- La tercera restricción garantiza que, contra cada **columna pura** del oponente, la utilidad sea al menos v. Por ende, si el oponente mezcla, la utilidad seguirá siendo $\geq v$.
- El objetivo es maximizar v.

3.2. Equivalencia con resolución mediante una variable de escalado

En muchos textos se introduce una variable "u" para eliminar la necesidad de v negativo, etc. Por ejemplo, a menudo se fuerza la matriz A a tener entradas no negativas sumándole un offset. Pero, a nivel conceptual, la formulación anterior ya da la idea.

3.3. Dual del Problema (para Jugador Columna)

El **dual** se interpreta como el problema de Jugador 2 que desea **minimizar** la utilidad de Jugador 1. El resultado de resolver el primal o el dual es el mismo valor v. Jugador 2 obtiene una mezcla $\mathbf{y}$ con la cual la utilidad no puede superar v.

---

4. Pseudocódigo del Algoritmo

Para resolver el **problema minmax** computacionalmente, típicamente se recurre a un **resolver de PL** (Programación Lineal). Sin embargo, a nivel de pseudocódigo, podemos dar una visión simplificada:


```
Entrada:
    - Matriz A (dimensión m x n) con valores (pueden ser positivos y/o
negativos).

Objetivo:
    - Hallar distribución x (x_i >= 0, sum x_i = 1)
    - Hallar valor v (escalar)
    - Tal que para cada columna j: sum_i( x_i * A[i,j] ) >= v
    - Maximizando v

Algoritmo (Bosquejo):

1. Normalizar (opcional):
   - Sea alpha = -min(A); (el valor más negativo de la matriz)
   - Construir A' = A + alpha (sumar alpha a cada entrada de A para
volverla no negativa).
     Esto a veces simplifica el tratamiento, pero no es obligatorio.

2. Crear variables y restricciones en un solver de PL:
   - Variables: x_1, x_2, ..., x_m (>=0), y la variable v.
   - Restricciones:
       R1: x_1 + x_2 + ... + x_m = 1
       R2: Para cada j en {1..n}:
             sum_i( x_i * A[i, j] ) >= v
       (Si se hizo normalización, adaptar la restricción para A'.)
   - Función Objetivo: Max v

3. Ejecutar un método de PL (por ejemplo, método símplex o interior-
```

```
point) sobre dichas restricciones.

4. Obtener solución:
   - (x_1^*, ..., x_m^*)  y valor v^*
   - Ese v^* es el "valor del juego" para el Jugador Fila.
   - (x_1^*,...,x_m^*) es la mezcla óptima del Jugador Fila.

5. (Opcional) Resolver el dual para obtener la mezcla y del Jugador
Columna:
   - Mín v
   - sujeta a: sum_j( y_j ) = 1, y_j >= 0, y "sum_j( y_j * A[i,j] ) <= v
para cada i"
   - O recuperar la mezcla dual con el método que devuelva los
multiplicadores duales.

Salida:
   - La estrategia mixta óptima de Jugador Fila (x^*)
   - La estrategia mixta óptima de Jugador Columna (y^*)
   - Valor del juego v^*
```

Nota: En la práctica, basta con usar **cualquier** solver de PL estándar (por ejemplo, *simplex*, *interior point*, *branch & bound* en caso de variables binarias, etc.) y plantear el problema con las restricciones indicadas.

---

5. Complejidad Computacional

5.1. Resolución de PL en general

- La **programación lineal** puede resolverse en **tiempo polinomial** en la *tamaño* de la entrada (con métodos de punto interior o el método del elipsoide).
- El método símplex, en el peor caso, puede tener complejidad exponencial, pero en la práctica suele ser muy eficiente.
- Para una matriz de **dimensión $m \times n$**, los **números** en la matriz influyen en la *longitud* de la descripción. Si la matriz tiene coeficientes que pueden representarse en L bits, las técnicas de punto interior usualmente resuelven en **$\mathrm{O}(\mathrm{poly}(m+n,,L))$**.

5.2. Casos específicos

1. **Juegos pequeños (m, n muy reducidos)**:

   - Podemos resolverlos directamente con la fórmula de PL y un solver genérico en un tiempo muy manejable.

2. **Juegos grandes (m, n grandes)**:

   - La dimensión de la matriz puede hacer que un método de PL con $\mathrm{O}(mn)$ restricciones (o variables) sea costoso, pero *sigue siendo polinomial* en $(m+n)$.
   - En problemas con m, n en el orden de millones, se requieren métodos más especializados (algoritmos de aproximación, descomposición, etc.).

3. **Juegos con estructura especial** (p.ej. "juegos matriciales esparsos" o "juegos en grafos"):
    - A veces podemos explotar la estructura para acelerar la resolución (por ejemplo, usando *column generation*, *row generation*, etc.).

5.3. Interpretación general

- **Polinomial** en el sentido de la **Teoría de la Complejidad**: existen algoritmos (elipsoide, interior point) que garantizan que el número de pasos es polinómico con respecto al número de bits necesarios para describir la instancia.
- En **casos prácticos**, el método símplex suele bastar: su **rendimiento medio** es muy bueno, a pesar de su peor caso exponencial.

---

6. Ejemplo Concreto Paso a Paso

Para ilustrar, tomemos un **juego de suma cero** con la siguiente **matriz de pagos** A para Jugador Fila (2 filas, 3 columnas), algo sencillo:

$$
A = \begin{pmatrix}
1 & -1 & 2 \\
-2 & 1 & 0
\end{pmatrix}
$$

- Jugador Fila (1) elige $\{F_1, F_2\}$.
- Jugador Columna (2) elige $\{C_1, C_2, C_3\}$.
- Si Jugador 1 elige fila 1 y Jugador 2 columna 3, Jugador 1 gana 2, etc.

6.1. Formulación PL para Jugador Fila

Variables:

- $x_1, x_2 \ge 0$, $\; x_1 + x_2 = 1$ (mezcla de Jugador 1).
- v = valor de juego a maximizar.

Restricciones (una por cada columna):

- Columna 1: $1\cdot x_1 + (-2)\cdot x_2 \ge v$.
- Columna 2: $(-1)\cdot x_1 + 1\cdot x_2 \ge v$.
- Columna 3: $2\cdot x_1 + 0 \cdot x_2 \ge v$.

Función objetivo:

$$
\max \; v.
$$

6.2. Resolución Intuitiva / Analítica

Podemos resolverlo a mano:

1. $x_1 + x_2 = 1$.
2. Restricciones:
    - (1) $; x_1 - 2x_2 \ge v$.
    - (2) $-x_1 + x_2 \ge v$.
    - (3) $; 2x_1 \ge v$.

Podemos intentar "igualar" estas expresiones en un *candidato* de equilibrio, con la idea de que en la mezcla óptima, Jugador Fila (1) suele "activarle" la misma utilidad contra las columnas relevantes. Veamos:

- De (3) inferimos: $v \le 2x_1$.
- También $;x_2 = 1 - x_1.$

Supongamos que la columna 1 y la columna 2 "importan" en la mezcla, es decir, que en equilibrio deben rendir lo mismo que la columna 3.

- De (1): $x_1 - 2x_2 = x_1 - 2(1 - x_1) = x_1 - 2 + 2x_1 = 3x_1 - 2.$
  Queremos $3x_1 - 2 = v.$
- De (2): $-x_1 + x_2 = -x_1 + (1 - x_1) = 1 - 2x_1.$
  Queremos $1 - 2x_1 = v.$
- De (3): $2x_1 = v.$

Si todas se igualan, tenemos:
$
3x_1 - 2 = 1 - 2x_1 = 2x_1.
$
Resolver:

- De $3x_1 - 2 = 2x_1$ se deduce $x_1 = 2$. (Pero ojo, $x_1\le 1$ porque es una probabilidad, así que esto no es posible).
- De $1 - 2x_1 = 2x_1$ se deduce $1 = 4x_1$ $\Rightarrow$ $x_1 = 0.25$.
    - Entonces $v = 2x_1 = 2(0.25) = 0.5.$

Probemos con $x_1=0.25$. Verifiquemos la restricción (1):

- $3(0.25) - 2 = 0.75 - 2 = -1.25$.
  Eso daría $,-1.25$, que no es ≥ 0.5. Contradice la restricción (1).

Conclusión: no pueden estar las tres restricciones "activas" a la vez. Quizá la columna 1 no sea "activa" en la mezcla.

Estrategia: Busquemos un *par de columnas* relevantes. A menudo, un método rápido es intentar "activar" *2 de las 3 restricciones* y ver qué ocurre.

- Si activamos la (2) y la (3):
  $
  \begin{cases}
  -x_1 + x_2 = v, \

2x_1 = v.
\end{cases}
$
Pero $x_2 = 1 - x_1$. Sustituyendo:
$
-(x_1) + (1 - x_1) = 1 - 2x_1 = v, \quad 2x_1 = v.
$
Entonces $1 - 2x_1 = 2x_1 \Rightarrow 4x_1 = 1 \Rightarrow x_1 = 0.25$. Y $v = 2x_1 = 0.5$.

  - Revisamos la (1): $,x_1 - 2x_2 = 0.25 - 2(0.75) = 0.25 - 1.5 = -1.25$.
  - Efectivamente, $-1.25 \ge 0.5$ no se cumple, pero no es necesaria la igualdad si (1) \textit{no} es la restricción que define v. Más bien, $\mathbf{x}$ debe satisfacer $\mathbf{x}^\top A e_j \ge v$ para **todas** j; en este caso no lo cumple.
  - Conclusión: esta $\mathbf{x}$ no es factible.

- Si activamos la (1) y la (3):
  $
  \begin{cases}
  x_1 - 2x_2 = v, \
  2x_1 = v.
  \end{cases}
  $
  Con $x_2 = 1 - x_1$:
  $
  x_1 - 2(1 - x_1) = x_1 - 2 + 2x_1 = 3x_1 - 2 = v,
  \quad 2x_1 = v.
  $
  De $2x_1 = 3x_1 - 2$ $\Rightarrow$ $3x_1 - 2x_1 = 2$ $\Rightarrow$ $x_1 = 2$.

  - De nuevo, no válido como probabilidad (>1).

- Si activamos la (1) y la (2):
  $
  \begin{cases}
  x_1 - 2x_2 = v,\
  -x_1 + x_2 = v.
  \end{cases}
  $
  Substituyendo $x_2 = 1 - x_1$:

  - (1): $3x_1 - 2 = v$.
  - (2): $-x_1 + (1 - x_1) = 1 - 2x_1 = v.$
    Igualamos: $3x_1 - 2 = 1 - 2x_1 \Rightarrow 5x_1 = 3 \Rightarrow x_1 = 0.6.$
    $;v = 1 - 2(0.6) = 1 - 1.2 = -0.2.$
    Revisa la (3): $2x_1 = 1.2$. ¿Es ≥ -0.2? Sí, es $\ge$ 0. Y no contradice.
  - De hecho, la restricción (3) pide $2(0.6) = 1.2 \ge v$, y con $v=-0.2$ se cumple de sobra.

PROF

Entonces $\mathbf{x} = (0.6, 0.4)$, $v=-0.2$. ¿Es factible de verdad?

- Revisemos cada columna:
  - Col1: $x_1(1) + x_2(-2) = 0.6 * 1 + 0.4 * -2 = 0.6 - 0.8 = -0.2 \ge v$?
    $; v=-0.2$. SÍ, se cumple con igualdad.
  - Col2: $0.6*(-1) + 0.4*(1) = -0.6 + 0.4 = -0.2 \ge v$?
    Igualdad también, se cumple.
  - Col3: $0.6*2 + 0.4*0 = 1.2 \ge -0.2$. Cumple.

Conclusión: $,(x_1, x_2) = (0.6, 0.4)$, $v=-0.2$.
Esto significa que el **valor del juego** para Jugador 1 es -0.2. Con esa mezcla, Jugador 1 "asegura" un pago promedio de -0.2. (En un juego de suma cero, el Jugador 2 podrá forzarlo a no superar -0.2.)

Podemos resolver el **dual** para encontrar la mezcla de Jugador Columna $\mathbf{y}$. Pero ya vemos que la estrategia y el valor -0.2 satisfacen las restricciones.

6.3. Interpretación

- Jugador 1, si mezcla filas con $(0.6, 0.4)$, obtiene en promedio -0.2, asumiendo que el Jugador 2 también juega de forma óptima.
- Significa que el juego, con esta matriz, **favorece** al Jugador 2 (valor < 0).
- Esto **coincidiría** con lo que un solver de PL daría: (x^* = (0.6, 0.4), v^* = -0.2).

Síntesis Final

- Para **juegos de suma cero**, la solución (o equilibrio de Nash) se encuentra resolviendo un **problema de programación lineal** (minimax / maximin).
- **Complejidad**: con PL, se puede hacer en tiempo polinomial en el tamaño de la matriz y la precisión de los datos. Para **grandes dimensiones**, se requieren métodos avanzados o heurísticas.
- El **algoritmo** es conceptualmente:
  1. Plantear las restricciones que garantizan la utilidad $\ge v$ (o $\le v$).
  2. **Maximizar o minimizar** ese valor.
  3. Usar un **solver de PL** (por ejemplo, símplex, interior-point).
- El **ejemplo** ilustra cómo se establecen las condiciones en la práctica y cómo se resuelve analíticamente para juegos pequeños.

De esta forma, tanto teóricamente (por el Teorema Minimax) como computacionalmente (Programación Lineal), los **juegos de suma cero** se resuelven de manera elegante y en tiempo **polinomial** respecto al tamaño de la descripción del juego.