7 PROBLEMAS DE JUEGOS

7.1 TEORIA DE JUEGOS

La teoría de juegos es una rama de la Economía (Matemáticas), corresponde a un entorno multiagente donde los agentes interactúan de forma cooperativa o competitiva. En el entorno aparece la contingencia como un factor adicional.

7.1.1 JUEGOS DE SUMA CERO

También llamados de suma constante, son los juegos más simples y tienen las siguientes características:

- Participan dos oponentes compitiendo que juegan por turnos alternos. Corresponden a agentes competitivos.
- Responde a un entorno:
    - Determinístico.
    - Totalmente observable.
- Lo que gana uno, lo pierde el otro.

Un juego de suma cero es denominado de esa manera, por cuanto lo que pierde un adversario lo gana el otro, se implementa a través de una función de utilidad que lo que a un jugador le da positivo al otro le da negativo. Por ejemplo a un jugador le puede dar 100 y al otro -100, la suma de estos valores dará 0, de ahí su nombre.

7.1.2 ALGORITMOS DE BUSQUEDA APLICADOS A JUEGOS DE SUMA CERO

Los juegos en su generalidad presentan algunas limitaciones para ser utilizados por algoritmos tradicionales de búsqueda como ser:

a) Alto factor de ramificación, mucha profundidad y se exige el óptimo. En función a numero de posibles jugadas cada juego es intratable por la mayoría de algoritmos de búsqueda. Como ejemplo podemos mencionar:
    - El 3 en raya: 9! = 362.880 nodos.
    - Ajedrez 10^{40} Nodos.
b) Tiempo limitado, falta de información y elementos de azar. Por lo que el entorno ya no será determinista lo que dará origen a la aparición de contingencias.

Este tipo de aspectos requieren de la implementación de una buena heurística, un criterio adecuado de utilidad y tecnicas eficientes de poda.

Por lo establecido, los algoritmos basados primero en anchura, no servirán para abordar este tipo de problemas, por lo cual se partirá de algoritmos basados en primero en profundidad. Primero se considerará asumir las restricciones del inciso a, para ir posteriormente resolviendo los siguientes incisos, con diferentes técnicas y estrategias.

Antes de empezar a tratar un determinado juego a partir de un algoritmo especifico, es importante definir suficientemente el problema que no varía mucho de los problemas de búsqueda revidados anteriormente, con la diferencia que se introducen los jugadores y el la función de utilidad.

Para definir un problema de tipo juego se requiere:

- Estado Inicial, configuración de inicio del juego, generalmente el nodo a partir del cual se inicia todo el procedimiento de búsqueda.
- Jugadores, el número de jugadores y quién tiene el turno. En problemas de suma cero son solo 2 jugadores.
- Acciones, la lista de movimientos legales para cada jugador en un estado. La lista de acciones completa que se pueden hacer por cada jugador y de ahí se tiene que ver cuales son realizable en ese momento especifico.
- Resultados, el modelo de transición. Estando en un estado A, aplicando una acción se conseguirá un estado B. No cambia en relación a lo revisado en los algoritmos de búsqueda revisados anteriormente
- Objetivos: estados donde el juego termina. Lista de posibles estados donde el juego termine. Esto no considera necesariamente que un jugador gane, puede darse el caso de un empate o una situación que impida que el otro jugador pueda continuar jugando.
- Utilidad: valor numérico de un estado objetivo para un jugador. En principio se debe dar un valor de utilidad a lo que son los estados objetivos, para posteriormente ir dando valores a los estados intermedios.

7.2 MINIMAX

Es el algoritmo mas sencillo e inocente, por que se puede rastrear todas las posibilidades del juego, tiene las siguientes características:

- Se basa:
  - En el algoritmo de búsqueda primero en profundidad.
  - En la búsqueda Y-O (AND-OR).
- Contingencia: por cada acción del jugador (nodo O) se debe analizar todas las posibles acciones del otro jugador (nodo Y).

Las características de juegos basados en minimax son las siguientes:

- Objetivo: conseguir la mejor estrategia, la secuencia de acciones óptima. El algoritmo debe dar la solución óptima.
- Jugadores: según utilidad (uno maximiza o mínima algo con la función de utilidad):
  - Max: agente inteligente, maximiza.
  - Min: contrincante, minimiza.
- Complejidad: exponencial $O(b^m)$, no se puede reducir a polinómica.

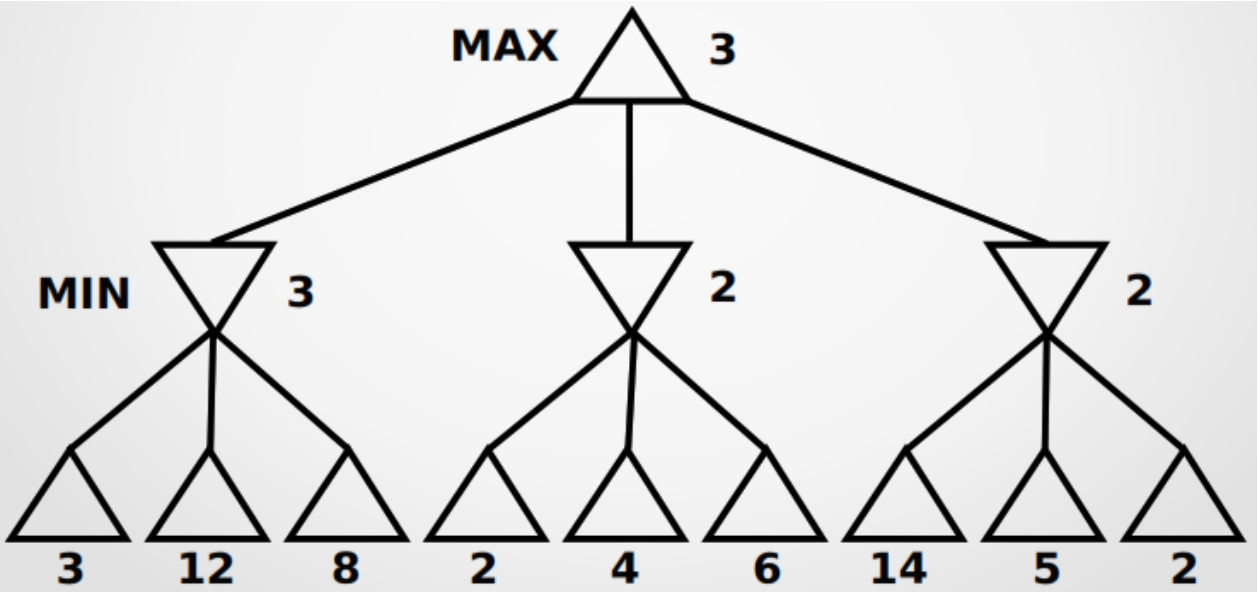
Cada nodo va a tener un valor, para calcular ese valor se utilizará la función de utilidad, en principio en el algoritmo minimax, se supondrá que solo los nodos finales, aquellos en los que un u otro agente gana o empata, considerados nodos finales, los cuales contendrán un valor, el valor de los nodos intermedios será calculado a partir de estos. Para mejorar la comprensión se realiza la siguiente definición:

El valor MiniMax depende del estado MiniMax(estado) :

- Es-Objetivo(estado) → Utilidad(estado)

- Jugador(estado) = MAX
  $max_{acción}$[MiniMax(Resultado(estado, acción))]
- Jugador(estado) = MIN
  $min_{acción}$[MiniMax(Resultado(estado, acción))]

Para mejorar la comprensión la dinámica que emplea el MiniMax, utilizaremos un ejemplo. Para comprender la figura siguiente, MAX corresponde con nosotros(Agente inteligente) y MIN con el adversario. Cada uno tiene un turno, nosostros realizamos algo y el otro contesta y ahí termina el juego. Cada uno puede elegir una de tres acciones.



MAX Buscara conseguir el mayor valor y MIN buscara que MAX consiga el menor valor. Si MAX elige el primer nodo partiendo de la izquierda, MIN tendría 3 opciones (3, 12 y 8) y elegirá el menor valor de los tres para que MAX no pueda obtener un valor de 12 o de 8, por que ese es el objetivo de MIN, elegir el nodo que minimice el valor para MAX, si MAX eligiera el nodo del medio o de la derecha, podría lograr uno de los mejores valores de utilidad como es el caso de la derecha con el valor 14, pero no olvidemos que esa es la aspiración, pero MIN elegirá la peor alternativa, es decir de los valores del nodo de la derecha (14, 5, 2) MIN elegirá 2, por cuanto de las tres alternativas la que da un mayor valor de los menores que elegirá MIN el primer nodo es el más adecuado porque MIN elegirá 3 que es mayor a 2 si se elegiría cualquiera de los otros dos nodos. Las tres opciones son malas, pero la primera es la menos mala. Como a los nodos no se les puede asignar directamente un valor de utilidad, se debe aplicar un criterio que permita realizar esto de una manera artificiosa.

7.2.1 PSEUDOCODIGO ALGORITMO MINIMAX

```
función MINIMAX(problema) devuelve acción
  mejor-accion   ← nulo
  mejor-utilidad ← 0
  inicial ← problema.ESTADO-INICIAL
  por cada accion en problema.ACCIONES(inicial) hacer
    resultado ← problema.RESULTADO(inicial, accion)
    utilidad  ← VALOR-MIN(problema, resultado)
    si utilidad > mejor-utilidad entonces
      mejor-accion   ← accion
      mejor-utilidad ← utilidad
  devolver mejor-accion
```

Devuelve una acción a realizar, la variable mejor acción que se haya encontrado, que será la que se devuelva, mejor utilidad corresponde a la mejor acción , variable inicial del problema (nodo inicial), suponiendo que tenemos el turno, se revisan todas las acciones y se revisa el resultado, aplicando al estado inicial, calculando la utilidad, suponiendo que tenemos el turno MAX, llamamos a MIN pasandole el problema y el resultado, para que cree un árbol y nos devuelva una utilidad y luego revisamos si la utilidad es mayor a la que se tiene, se reasigna la nueva mejor acción y utilidad, cuando se termina con todas las acciones se retorna la mejor acción. VALOR-MIN es una función recursiva. Que pasa si estamos con el turno de MIN, todo seria igual excepto que en lugar de llamar a VALOR-MIN se llama a VALOR-MAX, cambiando la condición de utilidad > mejor-utilidad por utilidad < mejor-valor. Ahora se presenta el pseudocódigo de VALOR-MIN:

```
funcion VALOR-MIN(problema, estado)
devuelve utilidad
  si problema.ES-OBJETIVO(estado) entonces
    devolver problema.UTILIDAD(estado)
  menor-valor ← +infinito
  por cada accion en problema.ACCIONES(estado) hacer
    resultado ← problema.RESULTADO(estado, accion)
    utilidad ← VALOR-MAX(problema, resultado)
    si utilidad < menor-valor entonces
      menor-valor ← utilidad
  devolver menor-valor
```

Y este es el pseudocódigo de VALOR-MAX:

```
funcion VALOR-MAX(problema, estado)
devuelve utilidad
  si problema.ES-OBJETIVO(estado) entonces
    devolver problema.UTILIDAD(estado)
  mayor-valor ← –infinito
  por cada accion en problema.ACCIONES(estado) hacer
    resultado ← problema.RESULTADO(estado, accion)
    utilidad ← VALOR-MIN(problema, resultado)
    si utilidad > mayor-valor entonces
      mayor-valor ← utilidad
  devolver mayor-valor
```

VALOR-MAX llama a VALOR-MIN y este llama a VALOR-MAX recursivamente, si se tendría tiempo límite, por la estructura del algoritmo, siempre habrá algún valor para devolver aunque no sea el mejor, pero no será malo. Para esto se debe adaptar el código a la restricción de un tiempo límite.

En las dos funciones se incluye la verificación de cumplimiento del objetivo, que es la condición de retorno de la recursividad, luego se define una variable mayor-valor o menor-valor con limites de mas y menos infinito respectivamente para compararlos con las funciones de utilidad. Luego se revisa cada una de las acciones que se dan en un determinado estado y si la función es VALOR-MIN, llama a la función VALOR-MAX en caso contrario VALOR-MAX, llama a VALOR-MIN, hasta que se cumpla la condición de llegar al objetivo. Después de conseguir el mayor o menor valor en función al turno del jugador que se establezca, se elegirá el valor mas conveniente y se esperara que el jugador haga su jugada y se ejecute nuevamente el MiniMax.

Este es el algoritmo base, sobre el que se aplican varias mejoras, o se agregan más complejidades.

7.2.2 MULTIPLES JUGADORES

En el caso de que el algoritmo MiniMax considere varios jugadores, es decir más de dos, n jugadores, los elementos que se deben considerar son:

- La función de utilidad en lugar de devolver un único valor debe devolver una tupla de valores: (j_1, j_2, …, j_n). (Las j son valores de utilidad para cada uno de los jugadores. La suma de todos debe dar cero.
- Turno: el jugador escoge, de todas las tuplas de los estados hijos, la que tenga mayor valor para él. La elección estará en función al turno.
- Complicaciones, aparecen alianzas, acuerdos, cooperaciones, entre otras temporales con propósitos transitorios, se modela con (POMDP). Emerge Cooperación, si la función de utilidad es la misma, esta puede ser voluntaria o involuntaria. Todo esto no corresponde a esta parte de la asignatura.
- No Suma Cero, se modela de la misma manera, con tuplas de n valores.