Curso: Inteligencia Artificial

Una simple y eficiente implementación del algoritmo Minimax

Resumen—**En el contexto de juegos entre dos adversarios de turno alternativo, el algoritmo minimax, consiste en la elección del mejor movimiento para un jugador, suponiendo que el contrincante escogerá uno que lo pueda perjudicar.**

Para escoger la mejor opción este algoritmo realiza un árbol de búsqueda, con todos los posibles movimientos, para luego recorrer todo el árbol de soluciones del juego a partir de un estado dado.

El algoritmo de búsqueda $\alpha - \beta$ representa una mejora al método minimax, pues se encarga de identificar subárboles y ramas del árbol de búsqueda que no van a influir en la decisión final. Para ello utiliza dos variables a lo largo del proceso de búsqueda en profundidad: α y β.

En este trabajo se presentará una implementación del algoritmo minimax y $\alpha - \beta$, utilizando Prolog, teniendo como ejemplo de pruebas el juego tic-tac-toe. Para ello hemos utilizado las ideas dadas en el libro de Rusell y Norvig [1], Nils J. Nilsson [4] y las implementaciones dadas de Stéphane Lens.

Las pruebas desarrolladas, indican que el algoritmo $\alpha - \beta$, todavía tiene que buscar en todos los caminos, hasta los estados terminales, para una parte del espacio de búsqueda. Esto no es realmente práctico, ya que los movimientos deben hacerse en una cantidad apropiada de tiempo, por ello se necesita un cambio en los programas.

Como dice el artículo de Shannon [5], estos deberían cortar la búsqueda antes y entonces aplicar una función heurística a los estados, convirtiendo, efectivamente, los nodos no terminales en hojas terminales.

El libro de Sterling y Shapiro [2] proporciona un ejemplo para el juego de Kalah. La complejidad de Kalah aprovecha mejor la búsqueda $\alpha - \beta$ utilizando estos cambios.

Palabras clave:—**Juegos con adversarios, minimax, búsqueda $\alpha - \beta$.**

I. Introducción

Existen diversas estrategias de búsqueda para encontrar soluciones a problemas que se pueden expresar por medio de un espacio de búsqueda con estructura en forma de árbol o de grafo. La teoría de juegos unipersonales es una de las fuentes más exitosas de problemas que se pueden resolver con estos procedimientos .

Pese a ese éxito, es difícil aplicar directamente estos procedimientos a **juegos con adversarios** en los que intervienen por lo menos dos jugadores. En este contexto, un método de decisión como el **Minimax** [1], [4] y las implementaciones que proporcionan una ganancia considerable como el método de $\alpha - \beta$ se utilizan para minimizar la pérdida máxima esperada en juegos con adversario y con información perfecta para maximizar la ganancia mínima esperada.

I-A. Formulación del problema

Para formular el problema, consideramos un juego con dos jugadores, llamados MAX y MIN.

MAX mueve primero, y luego intercambian turnos hasta que el juego se termina. Al final del juego, se conceden puntos al jugador ganador y penalizaciones al perdedor.

De acuerdo al libro de Russel y Norvig [1], un juego puede definirse formalmente como una clase de problemas de búsqueda con los siguientes componentes:

- El estado inicial, que incluye la posición del tablero e identifica al jugador que mueve.
- Una función sucesor, que devuelve una lista de pares (movimiento, estado), indicando un movimiento legal y el estado que resulta.
- Un test terminal, que determina cuándo se termina el juego. A los estados donde el juego se ha terminado se les llaman estados terminales.
- Una función utilidad, que da un valor numérico a los estados terminales.

El estado inicial y los movimientos legales para cada lado definen el árbol de juegos.

El **algoritmo minimax** calcula la decisión minimax del estado actual. Usa un cálculo simple recurrente de los valores minimax de cada estado sucesor.

Este método realiza una exploración primero en profundidad completa del árbol de juegos. Si la profundidad máxima del árbol es n, y hay b movimientos legales en cada punto, entonces la complejidad en tiempo del algoritmo minimax es $O(b^n)$, lo que para juegos reales estos costos de tiempos son poco prácticos.

El problema de la búsqueda minimax es que el número de estados que tiene que examinar es exponencial en el número de movimientos. Para reducir esta complejidad, una idea es la de cortar y eliminar partes del árbol de juegos. La técnica que utiliza esa idea es la que se conoce como $\alpha - \beta$.

La **búsqueda $\alpha - \beta$** es una técnica de búsqueda que reduce el número de nodos evaluados en un árbol de juego por el algoritmo Minimax. Para ello, trata de eliminar partes grandes del árbol que se va construyendo de forma que se devuelva el mismo movimiento que devolvería este, quitando ramas que se sepa que no van a influir en la decisión final.

$\alpha - \beta$ toma su nombre por el uso de dos parámetros que fijan ciertas cotas para el proceso de propagación hacia arriba que efectúa el algoritmo.

- α es el valor de la mejor opción hasta el momento a lo largo del camino para MAX, que representa, por tanto, una cota superior. El valor α representa la cota inferior del valor que puede asignarse en último término a un nodo maximizante.
- β es el valor de la mejor opción hasta el momento a lo largo del camino para MIN, que representa, por tanto, una cota inferior. El valor β representa la cota superior del valor que puede asignarse en último término a un nodo minimizante.

El cambio que se introduce en el minimax habitual es que en esta nueva versión se va actualizando el valor de los parámetros según se recorre el árbol. El método realizará el corte de las ramas restantes cuando el valor actual que se está examinando sea peor que el valor actual de α o β para MAX o MIN, respectivamente.

La eficacia de esta técnica es muy dependiente del orden en el que se examinan los sucesores. Según el artículo de Knuth y Moore [3], si los nodos están perfectamente ordenados, el número de nodos terminales considerados en una búsqueda de profundidad d con la técnica $\alpha - \beta$ es el doble de los nodos terminales considerados en una búsqueda de profundidad $d/2$ sin considerar $\alpha - \beta$.

Esto significa que en el caso perfecto, una búsqueda que haga uso de la búsqueda $\alpha - \beta$ proporciona ganancia considerable, ya que permite explorar con igual costo la mitad de los nodos finales de un árbol con el doble de profundidad.

II. Implementaciones en Prolog

Para indicar nuestras implementaciones en Prolog, primero utilizaremos el juego tic-tac-toe para ilustrar el método de minimax.

Supongamos que MAX marca con una equis y MIN marca con un círculo en el juego y que es el turno de MAX para jugar primero. Con un límite de profundidad 2, realizamos una búsqueda hasta que se generan todos los nodos en el nivel 2, y luego aplicamos una función de utilidad a las posiciones en esos nodos. Definimos una función de evaluación, $e(p)$, de una posición p como:

- Si p no es una posición ganadora para ninguno de los jugadores, entonces $e(p)$ = Número completo de filas, columnas o diagonales que están abiertas para MAX - Número completo de filas, columnas o diagonales que están abiertas para MIN.
- Si p es una victoria para MAX, $e(p) = \infty$ (denota un número muy grande).
- Si p es una victoria para MIN, $e(p) = -\infty$.

De estas deficiones hacemos uso de simetrías en la generación de posiciones del sucesor; por lo que se tendrá estados de juego que se consideran idénticos.

Al comienzo del juego, el factor de ramificación del árbol de tic-tac-toe se mantiene pequeño por simetrías; al final del juego, se mantiene pequeño por la cantidad de espacios abiertos. disponibles.

Las implementaciones en Prolog, basadas en los códigos de Gauthier Picard y de Stéphane Lens , que realizan estos procedimientos, de forma secuencial a lo explicado son:

- `minimax.pl`
- `tictactoe.pl`
- `tictactoegame.pl`

El empleo del procedimiento $\alpha - \beta$ siempre da como resultado encontrar un movimiento que sea lo 'suficiente bueno', como el movimiento que se habría encontrado con la misma profundidad que el método minimax, para hacer una correcta decisión.

De las definiciones de α y β, un valor actual, que debe ser encontrado está entre α y β. Si una posición muestra que tiene un valor que se encuentra fuera del intervalo α y β, es suficiente para saber que esa posición no está en la variación principal, sin conocer el valor exacto de esta posición. Solo sabemos el valor exacto de esa posición si ese valor está entre α y β.

Formalmente, podemos definir un valor 'suficientemente bueno' $e(p, \alpha, \beta)$ de una posición p, con respecto a α y β , como cualquier valor que satisfaga los siguientes requisitos:

$$\begin{aligned} e(p,\alpha,\beta) &< \alpha \quad \text{si} \quad e(p) < \alpha \\ e(p,\alpha,\beta) &= e(p) \quad \text{si} \quad \alpha \leq e(p) \leq \beta \\ e(p,\alpha,\beta) &> \beta \quad \text{si} \quad e(p) > \beta \end{aligned}$$

Podemos calcular el valor exacto $e(p)$ de una posición raíz p estableciendo los límites de la siguiente manera:

$$e(p, -\infty, +\infty) = e(p)$$

Las implementación en Prolog, para el método $\alpha - \beta$ es:

- `alfabeta.pl`

III. Pruebas

Para ejecutar las implementaciones se utilizaron dos entornos de Prolog: GNU Prolog para Linux y SWI-Prolog para Windows.

Para realizar las pruebas de las implementaciones, se utiliza el archivo `tictactoegame.pl`, que maneja una máquina de juego entre una jugador humano y una computadora y hace uso de los otros ficheros de Prolog. Cada una de los programas `tictactoe`, `minimax` o `alfabeta` e se llaman desde la cabecera de este archivo. Se utilizado en las pruebas el método `minimax`.

Después de haber lanzado este archivo en los dos entornos mencionados, se debe escribir:

```
?- jugar.
```

IV. Conclusiones

- Pese a que se no se han hecho pruebas de comparación con Prolog, si tenemos un árbol generado por el algoritmo minimax para tic-tac-toe, con una profundidad de 3, se puede demostrar que necesitamos generar 20 vértices menos y solo necesitamos generar 15 vértices si utilizamos el método $\alpha - \beta$. Se puede cortar el 57 % del árbol.
- Se ha realizado la implementación básica de los algoritmos minimax y $\alpha - \beta$ y se ha probado utilizando el juego del tic-tac-toe en una pequeña interfaz de juego para verificar su ejecución. Como prueba faltante no se han realizado los cambios de utilizar funciones de ponderado lineal cuando se corte la búsqueda, que realiza mejoras al método $\alpha - \beta$.
- De la investigación realizada para este trabajo, la ventaja real que trae el algoritmo $\alpha - \beta$ a juegos más complejos se deriva de su capacidad de cortar el árbol de búsqueda, mirando hacia delante la búsqueda de otros movimientos. El libro de Sterling y Shapiro [2] proporciona un ejemplo para el juego de Kalah. La complejidad de Kalah aprovecha mejor la búsqueda $\alpha - \beta$.

Referencias

[1] Stuart Rusell y Peter Norvig, Artificial Intelligence: A Modern Approach, Tercera edición, Pearson, 2009.

[2] Leon Sterling, Ehud Shapiro, The Art of PROLOG: Advanced Programming Techniques, (MIT Press, 1999.

[3] Donald E.Knuth, Ronald W.Moore, "An analysis of alpha-beta pruning", Artificial Intelligence 6: 293–326, 1975.

[4] Nils J. Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann Publishers, 1998.

[5] Claude E. Shannon, "Programming a Computer for Playing Chess", Philosophical Magazine, Ser. 7, Vol.41, No. 314, March 1950.