Memoria - Parcheckers

Autor: Lucas Hidalgo Herrera
Asignatura: Inteligencia Artificial
Grado: Doble Grado en Ingeniería Informática y Matemáticas

1. Introducción

A lo largo de la memoria, vamos a tratar las distintas mejoras del algoritmo *alfa-beta* implementadas para la práctica. Dichas mejoras son: *poda probabilística*, *poda con factor de ramificación* y *poda con ordenación.*

Además, como cabe esperar, antes de todo esto se explicará cómo se ha conseguido la implementación del algoritmo de poda *alfa-beta*.

Antes de terminar y aportar una tabla comparativa de resultados vamos a hablar un poco sobre la heurística elegida y por qué se ha elegido así.

2. Algoritmo de poda *alfa-beta*

2.1. Explicación de la obtención del algoritmo

Partimos del algoritmo *Min-max* proporcionado por los profesores en el tutorial de la asignatura. Este algoritmo ya nos da una base firme para obtener la poda en cuestión.

En un inicio, estamos considerando valores extremos para cada uno de los casos:

- *menosinf* en el caso de un nodo *MAX* pues buscamos maximizar nuestras ganancias.
- *masinf* en el caso de un nodo *MÍN* pues buscamos minimizar nuestras pérdidas.

Si recordamos cómo funciona la poda que buscamos implementar trabaja sobre un intervalo de extremos $[\alpha, \beta]$ donde el extremo α representa el valor mínimo que consideraremos para tomar la siguiente decisión y β como el valor máximo que consideraremos para tomar la siguiente decisión. Por tanto:

- En un nodo *MAX* buscaremos entre sus hijos y trataremos de tomar el máximo valor *minimax* de todos los hijos para considerarlo como nuestra cota inferior α; para así no tomar valores por debajo de este valor en futuras decisiones. Entonces, en estos nodos devolveremos el valor de α.
- En un nodo *MIN* buscaremos entre sus hijos y trataremos de tomar el valor *minimax* más pequeño de todos para considerarlo como nuestra cota superior β; para sí no tomar valores por encima de este valor en futuras decisiones. Entonces, en estos nodos devolveremos el valor de β.

Esta situación puede llevar a que $\alpha \leq \beta$, lo cual nos ocasiona una inconsistencia en el intervalo y lo interpretamos como una poda. Pero… ¿por qué se ocasiona una poda?

Contestando a la pregunta del párrafo anterior, si interpretamos esa desigualdad obtenida tenemos las siguientes situaciones:

- Como α es una cota inferior de las posibles decisiones que tomaremos, no elegiremos ningún valor que esté por debajo de él.
- Como β es una cota superior de las posibles decisiones que tomaremos, no elegiremos ningún valor que esté por encima de él.

En definitiva, no elegiremos ningún valor de las demás ramas que fuéramos a explorar sobre ese nodo y sólo desperdiciaríamos tiempo de ejecución. En estas situaciones devolveremos los valores que se explicaron en una situación normal para facilitar la legibilidad del código.

Volviendo ya a la implementación del algoritmo, esto se refleja en una simple comprobación de la poda en cada caso y en una actualización de los límites α y β.

2.2. Mi caso particular

Una vez que he realizado el algoritmo, tras una serie de pruebas cometí un error que, pese a no ser grave, fue difícil de depurar.

Mi algoritmo *alfa-beta* entraba en bucles infinitos en algunas ocasiones, o podaba cuando no debía de podar. Esto fue causado por la forma de pasar los argumentos de la función; me refiero a los límites superior e inferior, pues los pasaba por referencia. Haciendo que en cada llamada recursiva se manipulasen los límites del nodo raíz.

Esto ocasionaba que se obtuviesen resultados incorrectos.

3. Mejoras del algoritmo

He conseguido implementar tres de las cuatro mejoras que se nos han propuesto y son:

- Poda con profundidad dinámica basada en el factor de ramificación.
- Poda con ordenación de movimientos.
- Poda probabilística.
- Algoritmo definitivo del alumno.

3.1. Poda con profundidad dinámica basada en el factor de ramificación

Esta mejora se limita a calcular, cada vez que se ejecuta el algoritmo, cuál va a ser la profundidad máxima de exploración que se podrá alcanzar en sus hijos.

El cálculo de la nueva profundidad se basa en la función "*profundidad_siguiente*" donde se usan dos aspectos:

- Factor de reducción: es el encargado de modelar cuánto vamos a profundizar en el siguiente nivel en función de la ramificación que ha habido en este nivel, si la ramificación es alta, nos dará un factor pequeño. Probando constantes he considerado como ramificación alta el valor 8.
- Incremento: determina cuánto más vamos a profundizar como máximo, aquí entra en juego ese factor de reducción.

Para acabar, para que los límites sean razonables, no permitimos profundizar más de 20 niveles. Las constantes se han determinado para que el algoritmo sea bastante agresivo, a menores constantes menos agresivo será explorando en profundidad.

Cabe destacar que esta mejora es bastante notoria en cuanto a eficiencia; sobre todo en mi ordenador que es bastante lento con la poda *alfa-beta* normal.

3.2. Poda con ordenación de movimientos

Esta mejora es un poco traicionera, la intuición nos dice que, ordenando los nodos generados en cada nivel de profundidad nos dará un resultado mucho mejor que ordenando solo en el primer nivel de profundidad.

No obstante, esto no supone una diferencia tan notoria, ya que el tiempo que perdemos ordenando en proporción a la mejora de la ordenación es bastante grande. Además, debido a la restricción de número de nodos evaluados, será mejor no ordenar en todos los niveles pues en cada ordenación estamos evaluando nodos.

Por tanto, sólo ordenamos en el nivel de profundidad 0. Aún así es un algoritmo lento cuando el factor de ramificación es alto, es decir, cuando hay muchas jugadas por movimiento.

Cabe destacar que, probando esta mejora se consiguen buenos resultados que, en mi caso producen victorias que yo creía que no iban a ocurrir como el caso de la poda probabilística.

3.3. Poda probabilística

Esta mejora, de nuevo en eficiencia, consiste en contemplar el caso de la poda con resultados que probablemente sean poco positivos.

Se define un umbral de poda que trata de una constante de manera que, cuando se cumpla la ecuación $\beta - \alpha < umbral$, se realice una poda.

Esta mejora en eficiencia puede provocar peores resultados en la ejecución pues, si el umbral tomado no es adecuado estaríamos podando ramas del árbol que tendrían la solución.

La elección del umbral es algo que depende del rango de valores posibles que pueda tomar la función heurística; para el caso de *valoracionTest*, esta puede tomar valores dentro del intervalo $[-30, 30]$, entonces vamos a imponer un valor de umbral de $umbral = 3.5$.

Es una apuesta poco arriesgada que podará situaciones donde muy posiblemente acabaremos en desventaja.

3.4. Mi algoritmo de poda *alfa-beta*

He intentado realizar una mezcla de dos de los algoritmos de poda que hemos comentado anteriormente, concretamente el de poda con profundidad dinámica basada en el factor de ramificación y el de ordenación.

Sin embargo, la heurística que me otorga mejores resultados frente a los ninjas es con la poda simple.

Sinceramente, desconozco por qué ocurre eso, pero mi hipótesis es que el límite de nodos evaluados influye muy negativamente en las posibilidades de búsqueda de los algoritmos.

Por tanto, dependiendo de la heurística usaré una poda u otra.

4. Mi heurística

Antes de nada, debo indicar que la heurística que he considerado he tratado que siempre sea simétrica para que se cumpla lo visto en clase: “Lo que me beneficia debe perjudicar al contrincante y viceversa”.

Los aspectos que he considerado son:

- Tener todas las fichas en juego: en el parchís, tener una ficha en casa es perder oportunidades de comer, avanzar o incluso ganar. Si una ficha no está en juego no puede ganar la partida ni interactuar con el contrincante; por tanto, se ha tratado como algo perjudicial para ambos jugadores.
  Además, este aspecto también refleja las fichas que he comido y las que me han comido.

- Tener una ficha en meta: aunque puede parecer importante, lo que realmente es importante es estar cerca de la misma y en una casilla segura. Si ya nos encontramos cerca de la meta y en una casilla segura, muy probablemente estemos en el pasillo hacia la meta.

- Tener las fichas en el pasillo de la meta: como ya he dicho antes, estar en el pasillo de meta es una buena idea, si están ahí probablemente acaben entrando en meta.

- Tener una ficha bloqueada por una barrera: cuando llegamos a una barrera con alguna ficha, además de rebotar, podemos caer en una casilla poco segura aumentando nuestras posibilidades de ser comidos; por tanto, este aspecto afectará negativamente al jugador.

- Estar cerca de la meta: como ya se ha hablado en el punto anterior, una buena estrategia puede ser tratar de acercarse lo máximo posible a la meta, esto hará que ganemos cuanto antes si nadie nos lo impide.

- Estar en casilla segura: continuando con el punto anterior, “si nadie nos lo impide”, si no damos paso a que nadie nos impida llegar a la meta estando en casillas seguras el rival no tendrá posibilidad de pararnos; y si no impedimos al rival nosotros tampoco lo pararemos a él.

Con la última frase del párrafo anterior quiero llegar a la conclusión de que la elección de parámetros es arbitraria y difícil de realizar llegando a un ciclo; pues si no le doy peso a comerle fichas al rival tampoco lograré pararlo, pero si arriesgo mucho comiendo fichas puede que no tenga en cuenta llegar a la meta…

En definitiva, creo que tengo una idea de heurística bastante potente pero que con parámetros no optimizados no llegará a nada.

Centrándonos ya un poco más en el código, se puede ver que hay cuatro heurísticas implementadas. Dejo por aquí una breve descripción de cada una de ellas:

- Simple: aunque realmente de simple tiene poco, comenzó llamándose así porque fue una corrección de otra heurística que hice que era bastante compleja, controlaba la distancia máxima entre fichas del mismo jugador y aspectos parecidos. No obstante, es la que mejor resultado me da frente a los ninjas.
- TheBestInMyOpinion: en cómputo de partidas ganadas esto no es así, no es la que más partidas gana; de hecho, es la que más partidas pierde, de ahí su nombre sarcástico. Aparece en la práctica debido a que en el guión se nos recomienda dejar también los fracasos.
- GoForNinja1: es la heurística que más tiempo me ha llevado y peores resultados me ha dado, es simplemente prueba y error para ajustar parámetros, consigue vencer, sólo, al Ninja 1 como jugador 2, de una gran diferencia.
- AnotherTry: es idéntica a la mejor de todas, de hecho otorga resultados bastante parecidos, simplemente hay un cambio de parámetros que empeora un poco la solución reflejando la importancia de los mismos.

Por último, quiero hacer hincapié en otro aspecto que quería tener en cuenta para la heurística pero que finalmente consideré que la sobrecarga demasiado. Ese aspecto es tener una ficha amenazada por el rival. Considero que evitar, siquiera, la posibilidad de que el rival amenace nuestras fichas puede ganar partidas; sin embargo, en un tablero donde hay tantas fichas en juego y con los turnos tal y como están distribuidos es bastante improbable encontrar un hueco donde lleguemos a una posición segura sin estar en una casilla segura. Por tanto, consideré que era mejor no tenerlo; no obstante, la función “isInDanger” refleja que se ha realizado el intento.

5. Comparativa de resultados

Como comparativa de resultados vamos a realizar una tabla que refleje los enfrentamientos entre mis versiones de la poda *alfa-beta* con la heurística descrita anteriormente en el apartado 4.

Tabla comparativa con la heurística valoraciónTest

Ninjas	0		1		2		3		4		5	
Jugador	1	2	1	2	1	2	1	2	1	2	1	2
Alfa-beta	V	D	D	D	D	V	D	D	D	D	V	V
Ramificación	V	D	D	D	D	D	D	D	D	D	V	V
Ordenación	V	D	D	D	D	D	D	D	D	D	V	V
Probabilística	V	D	D	D	D	D	D	D	D	D	V	V
Definitiva	V	V	D	D	D	D	D	D	D	D	V	V

Tabla comparativa de las heurísticas

Ninjas	0		1		2		3		4		5	
Jugador	1	2	1	2	1	2	1	2	1	2	1	2
Simple	V	V	V	V	V	V	D	D	D	D	V	V
TBIMP	V	D	D	D	V	D	D	D	D	D	V	V
GoForNinja1	V	V	D	V	D	D	D	D	D	D	V	V
AnotherTry	V	V	V	V	V	V	D	D	D	D	V	V

Por tanto, gracias a las comparativas realizadas, como "*Simple*" y "*AnotherTry*" dan los mismos resultados, tomaré la primera de ellas.

6. Opiniones

Al igual que en la autoevaluación de la práctica de agentes, voy a incluir un punto donde hable sobre mi experiencia realizando esta práctica.

Por una parte, antes de nada, me gustaría decir que el software de la práctica está muy bien hecho, ha sido mucho más sencillo trabajar con todas las herramientas que se nos han proporcionado.

Por otra parte, es muy frustrante que la eficacia del algoritmo dependa de tantas constantes por determinar, he estado muchísimo tiempo intentando encontrar esas constantes óptimas. Sin embargo, lo máximo que he conseguido ha sido ganarle a tres de los seis ninjas que cuentan para evaluación.

No obstante, cuando ves funcionar, aunque sea un poco, una creación propia, es una satisfacción inmensa.