MEMORIA PRÁCTICA 4

INTELIGENCIA ARTIFICIAL

Nuria Cuaresma Saturio y Luis Cárabe Fernández-Pedraza G2301, P05

Pregunta C1. Explique los fundamentos, el razonamiento seguido, la bibliografía utilizada y las pruebas realizadas para la entrega de la/s función/es de evaluación entregada/s en el torneo.

Una primera aproximación al problema fue tener en cuenta ciertos parámetros que nosotros consideramos útiles para la valoración de una partida. Empezamos dándole valor a tener hoyos vacíos y a la posición de estos mismos, ya que considerábamos que si conseguíamos un tablero con hoyos vacíos en nuestra fila, aumentaban las posibilidades de conseguir un robo de semillas. No obstante, esto trajo consigo un problema grande, tendíamos a quedarnos sin semillas en el afán de conseguir más huecos vacíos, con lo cual, tuvimos que añadir otro parámetro, el número de semillas totales que teníamos en nuestra fila, así lográbamos un mejor equilibrio entre las posibilidades de robo y no desperdiciar semillas.

Por tanto, hasta ese momento teníamos dos mediciones con sus respectivas ponderaciones, el número de hoyos vacíos y el número de semillas, dedicamos unos días a cambiar los valores, dándonos cuenta de que era más importante retener semillas que conseguir vacíos. Al no conseguir la puntuación deseada en el ranking, probamos a añadir nuevos parámetros para tener en cuenta número de semillas en el kalaha, diferencia de fichas entre nuestro jugador y el oponente… Todos ellos tenían un factor de multiplicación y ya era pesado tener que probar valores manualmente, por tanto, procedimos a crear un generador automático (y aleatorio) de valores, funciona de la siguiente manera:

El fichero mancala11.cl fue modificado para que nos devolviera como resultado de la partida la resta de puntos del primer jugador con respecto al segundo (desactivando las ayudas del juego también), así si una partida imprime un número positivo querría decir que el primer jugador ha ganado al segundo, un número negativo significaría lo contrario. Este mismo fichero también tendría una función *ejecutar* que recibiría un jugador, entonces, dicha función utilizaría al jugador para disputar partidas contra el jugador aleatorio (**jdr-nmx-eval-aleatoria**), ya que representa un modelo próximo a lo que pueden ser los distintos jugadores del resto del ranking. Juega tantas partidas como le especifiquemos, tanto empezando uno como otro, para luego hacer una esperanza de los puntos de diferencia que nuestro jugador ha conseguido. Para que el resultado sea fiable, nuestros jugadores se han enfrentado al aleatorio en 5000 partidas (2500 teniendo el primer movimiento).

Tras esto, creamos un script .sh que recibirá los factores de multiplicación por argumento y gracias al comando sed, reemplazará la línea que se encarga de definirlos en nuestra heurística, es decir reemplazará *(defvar *parametros* '(1 2 3 4 5 ...))* por la misma declaración pero con los parámetros recibidos, tras esto, usará sbcl para ejecutar el código y recibir la salida de nuestra función *ejecutar*, que devolverá en forma de *echo*.

Ya solo faltó crear un fichero python .py que se encargara de generar números aleatorios con randint y ejecutar el script, guardando en un fichero el resultado. Con estos últimos pasos en el interior de un bucle, conseguimos crear un generador aleatorio de parámetros automático, aunque no muy eficiente, ya que no comprobamos si una

combinación de parámetros ya se ha evaluado anteriormente o no. No obstante, sí que ponemos un filtro para que todo jugador que no supere al *jugador regular* sea descartado automáticamente sin que llegue a jugar contra el aleatorio.

Aún con este generador, no obtuvimos los resultados esperados, lo que nos llevó a plantearnos un cambio de heurística. Razonando sobre el problema nos dimos cuenta de que lo que hacíamos hasta ahora, principalmente, se podía reducir a darle un valor a cada hoyo de nuestra fila, ya que en esto radica el pensamiento de dar importancia a tener muchas semillas o muchos hoyos vacíos, por tanto, pasamos a hacer un producto escalar, es decir, para cada hoyo de nuestra fila, asignábamos un factor de multiplicación, por tanto, ahora teníamos 6 parámetros con los que jugar en el generador.

El último añadido a la heurística fue uno tan obvio que nos sorprendimos de no haberlo tenido hasta el momento, una condición que comprobara si la partida había terminado en esa situación del tablero, en ese caso, comprobamos quién ha ganado, si es nuestro jugador, asignamos un valor alto a esa situación, un cero en caso contrario.

Con esta base, ya solo nos quedaba generar muchos valores y ordenar los ficheros de salida para quedarnos con los mejores. Los jugadores entregados son el resultado de muchas combinaciones posibles y muchas horas tras una ejecución lenta pero fiable.

Pregunta C2. El jugador llamado bueno sólo se distingue del regular en que expande un nivel más. ¿Por qué motivo no es capaz de ganar al regular? Sugiera y pruebe alguna solución que remedie este problema.

```
(defun f-eval-Bueno (estado)
  (if (juego-terminado-p estado)
     -50                    ;; Condicion especial de juego terminado
   ;; Devuelve el maximo del numero de fichas del lado enemigo menos el numero de propias
   (max-list (mapcar #'(lambda(x)
                         (- (suma-fila (estado-tablero x) (lado-contrario (estado-lado-sgte-jugador
x)))
                            (suma-fila (estado-tablero x) (estado-lado-sgte-jugador x))))
             (generar-sucesores estado)))))
```

Tras analizar el código del jugador bueno, comprobamos cuales son sus resultados contra el regular:

```
(partida 0 2 (list *jdr-nmx-Regular* *jdr-nmx-bueno*))
 Marcador:  Ju-Nmx-Regular 13 - 23 Ju-Nmx-Bueno

(partida 1 2 (list *jdr-nmx-Regular* *jdr-nmx-bueno*))
 Marcador:  Ju-Nmx-Regular 26 - 10 Ju-Nmx-Bueno
```

Con esto podemos ver que cuando el jugador regular empieza, gana el bueno, no obstante, cuando el bueno empieza, pierde. Ya que tienen la misma estrategia, pero simplemente el bueno expande un nivel más, quizás el problema radique en esa primera condición de *(if (juego-terminado-p estado) …)*, ya que comprueba si el estado a evaluar es uno en el cual la partida ha terminado, no obstante, no discrimina si esa partida es favorable a nuestro jugador o no, por tanto, poniendo otra condición más que determine esta situación, tenemos lo siguiente:

```
(defun f-eval-Bueno (estado)
  (if (juego-terminado-p estado)
     (if (> (cuenta-fichas (estado-tablero estado) (estado-lado-sgte-jugador estado) 0)
            (cuenta-fichas (estado-tablero estado) (lado-contrario (estado-lado-sgte-jugador
estado)) 0))
        9999 ; Si hemos ganado
       0) ; Si hemos perdido
;; Devuelve el maximo del numero de fichas del lado enemigo menos el numero de propias
    (max-list (mapcar #'(lambda(x)
                          (- (suma-fila (estado-tablero x) (lado-contrario (estado-lado-sgte-jugador
x)))
                       (suma-fila (estado-tablero x) (estado-lado-sgte-jugador x))))
```

Y esto arroja los siguientes resultados:

```
(partida 1 2 (list *jdr-nmx-Regular* *jdr-nmx-bueno*))
 Marcador:  Ju-Nmx-Regular 4 - 32 Ju-Nmx-Bueno

(partida 0 2 (list *jdr-nmx-Regular* *jdr-nmx-bueno*))
 Marcador:  Ju-Nmx-Regular 15 - 21 Ju-Nmx-Bueno
```

Podemos asegurar que ahora el jugador bueno gana en toda situación al regular.

Evaluación de la calidad de la Función de Evaluación:

Como explicamos en la pregunta C1, nuestros mejores jugadores (los cuales hacen el producto escalar) se diferencian del resto (en los que solo se tienen en cuenta ciertos parámetros) de una manera clara. Como ya dijimos, esto es debido a que evaluar la importancia de los parámetros no es más que asignar un valor de importancia a cada hoyo en nuestra fila, por ejemplo, una función que tiene en cuenta el número de hoyos vacíos más lejos del kalaha, para la heurística con el producto escalar puede ser dar un factor de multiplicación menor a dichos hoyos, o una heurística que prima el número de semillas en toda nuestra fila, para el producto escalar puede ser unos factores bastante altos en todos los hoyos. Es decir, juega con todos los parámetros implícitamente, los que nos podemos imaginar y los que no.

Por otra parte, son jugadores rápidos, que no expanden niveles y se limitan a hacer unas pocas sumas y multiplicaciones, con lo cual conseguimos una eficiencia más que suficiente para abordar el estado del tablero.