Análisis de UCT en el Reversi

Martín Arjovsky

Octubre 2012

1 Introducción

En este trabajo se analiza el rendimiento y la eficiencia del algoritmo Upper Confidence Bounds Applied to Trees (UCT), una variante de los algoritmos de Monte Carlo Tree Search en el popular juego Reversi. El trabajo permite evaluar como se desempeñan los algoritmos basados en Monte Carlo en conjunción con Aprendizaje por Refuerzos en juegos de mesa. También provee un breve análisis de variación de parametros y de cuándo son útiles los mismos en comparación con otros algoritmos.

2 El Juego

El reversi es un juego tradicional de mesa con un conjunto muy simple de reglas:

- Hay dos jugadores enfrentados, el blanco y el negro.
- El juego toma lugar en un tablero de 8x8 que empieza como se muestra en la figura 2.1.
- El negro mueve primero.
- Una jugada válida es aquella en que un disco es posicionado en un casillero vacío y deja una o mas piezas del oponente entre esta nueva pieza y otras del jugador. Esto puede tomar lugar de forma vertical, horizontal o diagonal y las piezas del oponente que quedan atrapadas son automaticamente convertidas a las del jugador.
- Si un jugador no tiene jugadas disponibles, entonces debe pasar.
- El juego termina cuando ambos jugadores tienen que pasar (por ejemplo, cuando el tablero está lleno).
- El ganador es el jugador que posee mas fichas en el momento de terminación del juego, y en caso de que ambos tengan igual cantidad de las mismas, termina en empate.

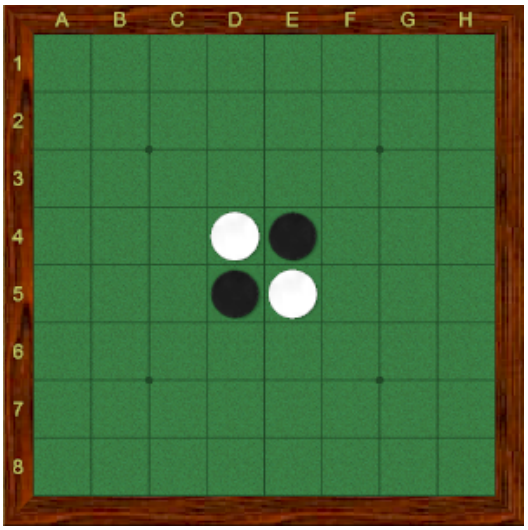

Figure 2.1: Posición Inicial

Se trata de un ambiente completamente observable, ya que todos los agentes conocen toda la información relevante del ambiente al momento de tomar sus decisiones. Es un juego multiagente y competitivo de suma cero, donde ambos jugadores están enfrentados y el beneficio de uno es estrictamente en detrimento del otro. El ambiente también es determinístico, ya que todos los agentes pueden predecir las consecuencias de sus acciones. Es estático, ya que el mismo no cambia mientras un agente está deliberando. Es secuencial, debido a que las decisiones del agente en el presente afectan decisiones futuras. Finalmente, se trata de un ambiente discreto, ya que la cantidad de estados es finita.

La estrategia en el Reversi tiene dos componentes básicos:

Estabilidad

Un disco se considera estable si no puede cambiar de dueño en lo que resta del partido. Las piezas que siempre tienen esta característica son los discos de las esquinas, debido a que no pueden nunca quedar atrapados. Más aún, una vez que un jugador posee una esquina, suele ser muy fácil hacerse de una sección de discos estables rodeando este disco. Es por esto que muchas veces el jugador que posea esquinas tiene una abrumador ventaja sobre su oponente.

Movilidad

Cuando un jugador tiene muchas movidas posibles se dice que tiene alta movilidad, y por el contrario, cuando no tiene o tiene pocas se dice que tiene baja movilidad. La movilidad es esencial en el Reversi, ya que tener mucha movilidad permite atacar al oponente de varias maneras distintas, y lo que es más importante, al suprimir la movilidad del oponente, uno lo

puede forzar a hacer malas decisiones que pueden llevarlo, por ejemplo, a perder esquinas.

A partir de estos dos componentes se derivan otros conceptos como el tempo (ganar una jugada extra al no dejar mover al otro y obligarlo a pasar) y la paridad, que consiste en poder forzar la posibilidad de poner la última pieza en un sector del juego, y así ganar control sobre la zona al conquistar las piezas del oponente en ese sector.

3 UCT

El algoritmo que se usó en este trabajo se denomina Upper Confidence Bounds Applied to Trees. El algoritmo estudia el arbol de decisiones hasta una cierta profundidad adelante en el juego, en donde (antes de mover) va a tratar de averiguar cuales son las situaciones mas prometedoras. Luego, una ves estudiadas, al algoritmo le gustaría ejecutar dentro de las jugadas inmediatas posibles, la mejor. Para evaluar las siguientes jugadas, como todo algoritmo de Monte Carlo Tree Search, hace cuatro cosas[1]:

1. Selección: Cuando un estado es encontrado en un arbol, la siguiente acción es elegida según los datos que se tienen almacenados, de alguna manera que balancee exploración y explotación. En general la estrategia trata de seleccionar la acción que dio mejores resultados hasta el momento (explotación). Por el otro lado, acciones menos prometedoras también deben ser estudiadas (exploración).

2. Expansión: Cuando el juego llega a un estado que no habia sido explorado, el estado es agregado como un nuevo nodo. Estos dos procesos se realizan hasta alcanzar la profundidad deseada.

3. Simulación: Por el resto del juego, las acciones son seleccionadas pseudoaleatoriamente hasta la finalización del mismo. Naturalmente, el peso que se le da a la selección de cada acción particular tiene un efecto significativo en el nivel de juego del agente. Si todas las acciones son elegidas con la misma probabilidad, entonces la estrategia jugada es generalmente débil y el nivel del agente es subóptimo. Debido a esto podemos usar heurísticas para darle mayor peso a acciones más prometedoras.

4. Backpropagation: Luego de haber llegado al final de la simulación, actualizamos cada nodo cuya acción haya tenido lugar durante el juego. La cantidad de visitas es incrementada y la cantidad de veces que se ganó o perdió es modificada según como haya salido la simulación.

UCT lo que hace es tomar como política de selección el algoritmo llamado UCB1 de intervalos de confianza. Al hacer esto, el algoritmo toma cada descenso en el arbol de jugadas como un problema individual de bandidos de K brazos. De esta manera tiende a balancear exploración y explotación eficientemente,

eligiendo siempre la acción que tenga el mayor valor posible en su intervalo de confianza. Para detalles sobre como hace esto ver el código de este trabajo o Gelly, S. & Silver, D. (2007).

Como heurísticas de simulación se tomaron dos relativamente sencillas. La primera, la que elige acciones al azar y la segunda, que asigna una prioridad a cada posición del tablero y luego elige un lugar con probabilidad directamente proporcional a la prioridad asignada. La siguiente figura muestra las prioridades que asigna la heurística en cada posición del tablero:

10000	-3000	1000	800	800	1000	-3000	10000
-3000	-5000	-450	-500	-500	-450	-5000	-3000
1000	-450	30	10	10	30	-450	1000
800	-500	10	50	50	10	-500	800
800	-500	10	50	50	10	-500	800
1000	-450	30	10	10	30	-450	1000
-3000	-5000	-450	-500	-500	-450	-5000	-3000
10000	-3000	1000	800	800	1000	-3000	10000

Como se puede observar, se privilegian por sobre todo las esquinas, que son las posiciones más estables del juego. En segundo lugar están los bordes, que proporcionan tanto estabilidad como facilidad de acceso a las esquinas. En tercer lugar el centro se prioriza mucho, debido a que su dominio porporciona gran movilidad. Finalmente, se evitan las zonas adyacentes a las esquinas, ya que poseer piezas en estas posiciones genera mucha vulnerabilidad de ataque a las mismas.

4 Experimentos y Análisis

Para estudiar el algoritmo y su funcionamiento, se realizaron varios experimentos. En primer lugar, se examinaron las curvas de aprendizaje de los Q values que iba computando el algoritmo a medida que hacía cada simulación en distintas situaciones, tanto de apertura, como de midgame y endgame. En segundo lugar, se evaluaron las decisiones que tomaba en posiciones estudiadas y fueron comparadas con las óptimas y subóptimas. Para todos los experimentos el algoritmo usó 2000 simulaciones, 6-ply search y la heurística de prioridades del tablero. La función de recompenza asigna $+1$ al ganador y -1 al perdedor. Finalmente, se estudió la performance del programa al ir variando distintos parámetros y heurísticas. Para ello, el algoritmo jugó contra si mismo variando estos factores.

El algoritmo recibe como parametros la situación completa del tablero y un jugador y devuelve una jugada. Los estados se representan con matrices de 8x8 que tienen ceros en las posiciones no jugadas y el número de un jugador en las posiciones ocupadas por el mismo. Las acciones se consideraron como números del 1 al 64, uno para cada posición del tablero. Para simular el ambiente se usaron dos funciones, una que dado un tablero y un jugador da los movimientos

válidos para el mismo y otra que dada una acción, un tablero y el jugador que la realiza, hace la acción y devuelve el tablero siguiente, la reward asociada, si es un estado final, el próximo jugador y sus movimientos válidos en el nuevo tablero. Para poder aumentar la cantidad de juegos simulados al enfrentar a los agentes con distintos parametros, estos juegos fueron paralelizadas y cada núcleo jugaba en un tablero distinto. La totalidad del algoritmo, el ambiente y los experimentos fueron programados en MATLAB.

4.1 Aprendizaje y tableros estudiados

Para comenzar el estudio de las curvas de aprendizaje y la toma de decisiones del algoritmo, primero vimos como aprendía a elegir su movimiento de apertura.

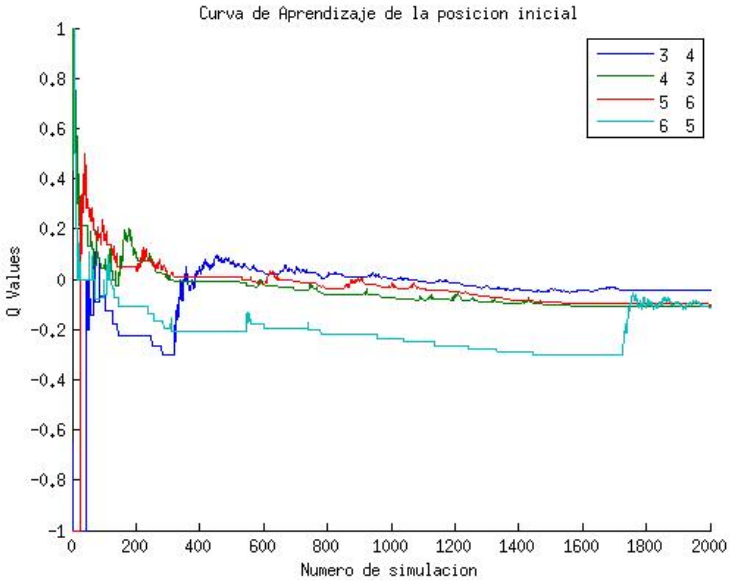

Figure 4.2: Curva de Aprendizaje para la posición inicial del tablero

En la figura 4.2 se observa como el algoritmo detecta que las posiciones iniciales tienen todas idéntico valor 0, lo que es consistente con la simetría de la posición y el hecho de que no hay una gran diferencia entre empezar como negro o blanco.

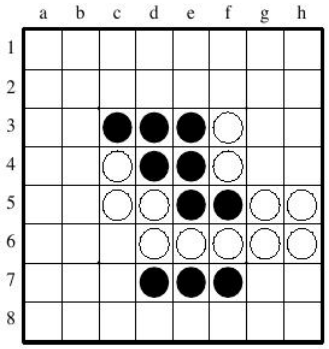

Figure 4.3: Tablero 1

En las figuras 4.3 y 4.4 se muestran el primer tablero de juego estudiado y la curva de aprendizaje respectivamente. El jugador que debe actuar es el negro.

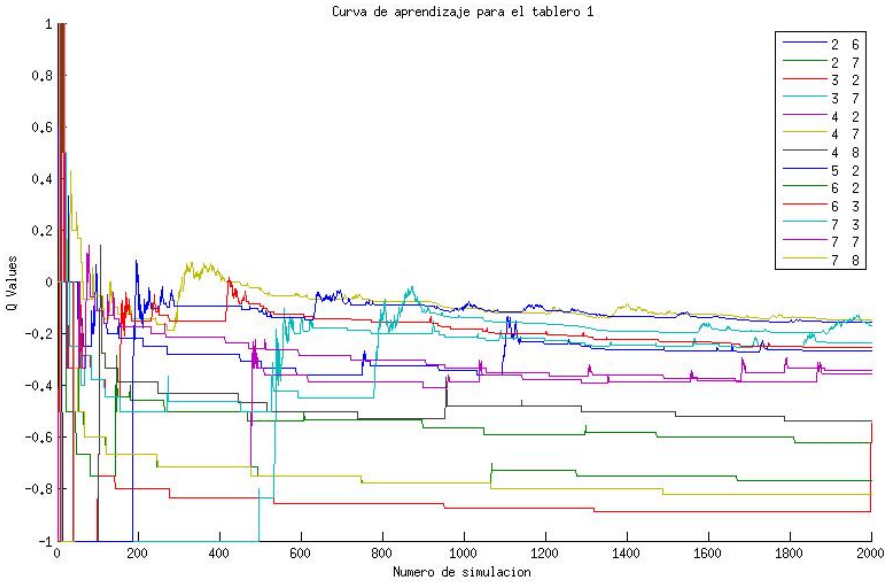

Figure 4.4: Curva de Aprendizaje para el tablero 1

La acción ejecutada por el algoritmo es g4 (4 7 en la leyenda). Esta jugada coincide con lo recomendado por la literatura.[3] Esto es debido a que esta jugada es lo que se considera "callada". Una jugada callada es aquella que no crea muchos discos de frontera, que son muy suceptibles a ser tomados y proporcionan movilidad al oponente. Además de ello, le saca la posibilidad al blanco de jugar g4, para quien sería una excelente jugada.

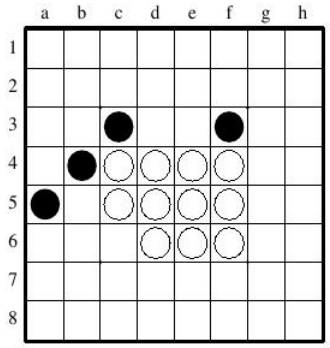

Figure 4.5: Tablero 2

En las figuras 4.5 y 4.6 se muestran el segundo tablero estudiado y su curva de aprendizaje. El jugador que debe actuar es el blanco.

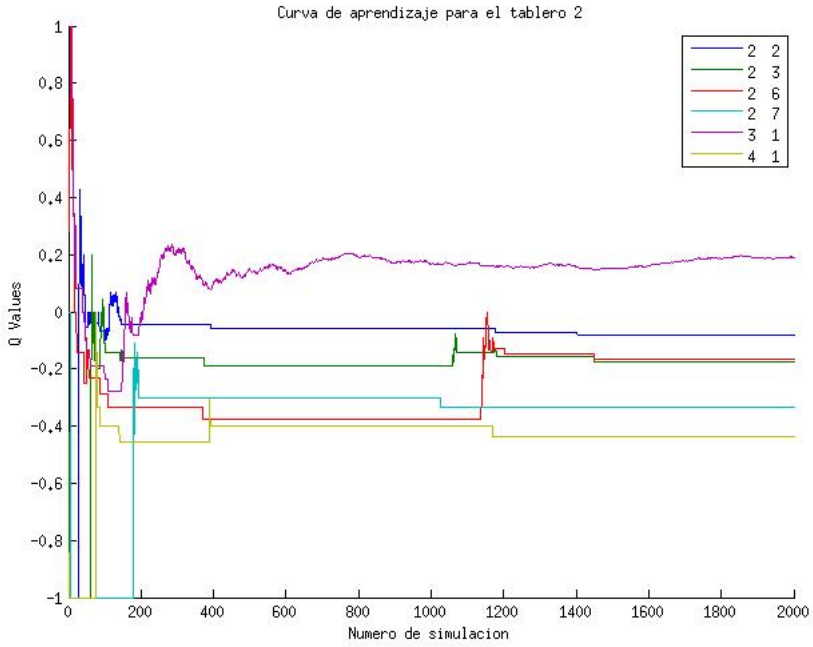

Figure 4.6: Curva de Aprendizaje para el tablero 2

Esta es una posición que ha ocurrido muchas veces en juego experto.[3] El blanco tiene un gran número de discos de frontera, pero tiene control del centro. El blanco debería jugar a3, dejando al negro con solo tres opciones, c6, f7 y g7,

de las cuales todas son fuertes (contrario a callada). Como se ve en la curva, esta es por lejos la opción preferida por el algoritmo.

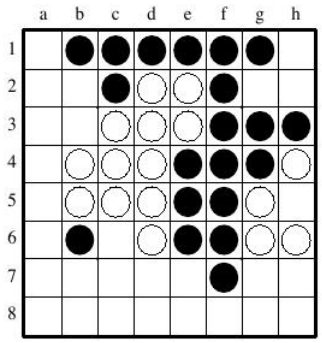

Figure 4.7: Tablero 3

En las figuras 4.7 y 4.8 se muestran el tercer tablero y su curva de aprendizaje. El jugador que debe actuar es el negro.

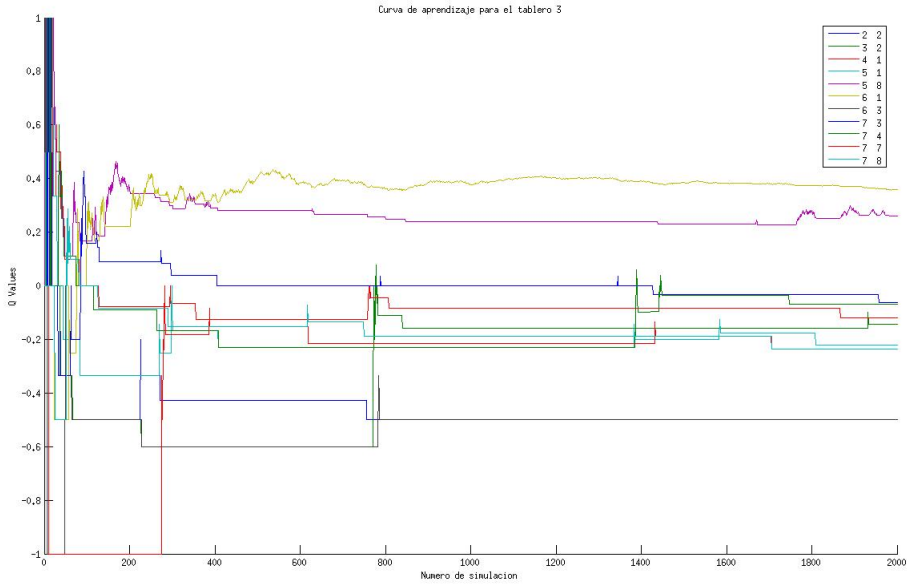

Figure 4.8: Curva de Aprendizaje para el tablero 3

En este caso, si el negro juega h5, daría vuelta el disco en g6 y el blanco tomaría el borde en h2, quedando con una jugada gratis a h7. La mejor jugada

del negro es a6,[3] sacándole el acceso a h5 al jugador blanco. Esta claramente es la jugada ejecutada por el algoritmo.

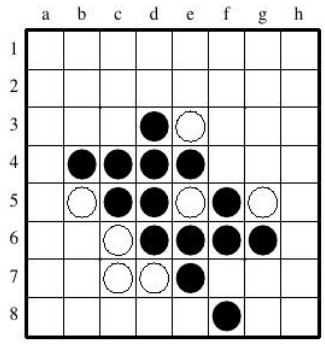

Figure 4.9: Tablero 4

En las figuras 4.9 y 4.10 se muestran el cuarto tablero y su curva de aprendizaje. El jugador que debe actuar es el blanco.

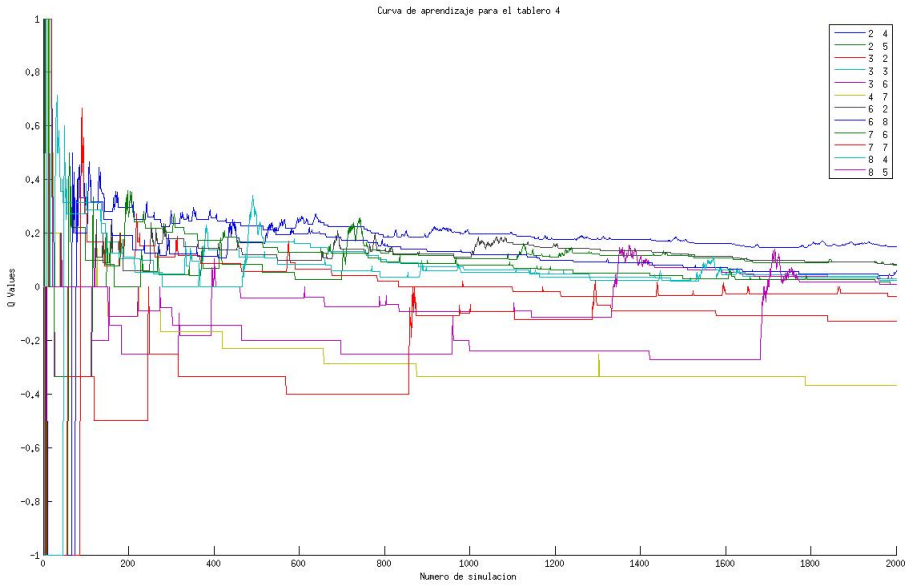

Figure 4.10: Curva de Aprendizaje para el tablero 4

El negro esta amenazando hacer una buena jugada a f4, así que el blanco debería jugar d2[3], dando vuelta el disco en d6. Hacer esto no solo le saca al

negro el acceso a f4 sino que ahora el blanco amenaza hacer una jugada callada a f4. Nuevamente se ve como el algoritmo detecta la mejor jugada.

4.2 Variación de Parámetros

Para estudiar el comportamiento del algoritmo se hizo competir al agente contra sí mismo variando los parámetros. Estos fueron, la cantidad de simulaciones, la profundidad de la búsqueda, la heurística de simulación y la funcion de recompenza. Para estudiar la variación en la cantidad de simulaciones y en la profundidad se realizaron 200 partidos para cada estudio, y en el estudio de las heurísticas 100, también para cada estudio. En todos los estudios, ambos jugadores jugaron la mitad de los partidos con blanco y la mitad como negro. A continuación se muestran los resultados.

Cantidad de simulaciones	Porcentaje de partidos ganados
2000	65%
1000	35%

Table 1: Resultados al variar la cantidad de simulaciones

Los estudios de la cantidad de simulaciones fueron hechos con 4-ply search, la heurística de prioridades del tablero y la función de recompenza que asigna $+1$ al ganador y -1 al perdedor. Como era de esperarse, al aumentar la cantidad de simulaciones, la performance del algoritmo aumentó dramáticamente.

Profundidad de la búsqueda	Porcentaje de partidos ganados
6-ply	57%
4-ply	43%

Table 2: Resultados al variar la profundidad

Los estudios de la profundiad fueron hechos con 2000 simulaciones, la heurística de prioridades del tablero y la función de recompenza que asigna $+1$ al ganador y -1 al perdedor. Al aumentar la profundidad de búsqueda, la performance del algoritmo aumentó significativamente. De todas maneras, es interesante notar que no tuvo un efecto tan marcado como al aumentar la cantidad de simulaciones.

Heurística de simulación	Porcentaje de partidos ganados
Prioridades del tablero	78%
Aleatoria	22%

Table 3: Resultados al variar la heurística de simulación

Los estudios sobre la heurística fueron hechos con 1000 simulaciones, 4-ply search y la función de recompenza que asigna $+1$ al ganador y -1 al perdedor. Al variar la heurística de simulación se notó una altísima diferencia de performance,

dando una clara victoria a la heurística que funciona en base a las prioridades del tablero. Esto refuerza la idea ya dicha de que el agente es altamente subóptimo con una heurística equiprobable.

Función de recompenza	Porcentaje de partidos ganados
$+1$ al ganador y -1 al perdedor	87%
Cantidad de piezas	13%

Table 4: Resultados al variar la heurística de simulación

Los estudios sobre la función de recompenza fueron hechos con 1000 simulaciones, 4-ply search y la heurística de prioridades de tablero. La función de recompenza de cantidad de piezas asigna $+n$ al ganador y $-n$ al perdedor, con $n =$ cantidad de piezas del ganador al terminar el partido. En estos experimentos se ve una brutal diferencia de performance, de donde se puede inferir que el hecho de ganar una partida con muchas o pocas piezas no es extremadamente representativo del desempeño durante el partido. También recuerda la importancia de encontrar una buena función de recompenza en algoritmos basados en aprendizaje por refuerzos, y una futura investigación en este tipo de algoritmos para el reversi podría centrarse en buscar una mejor función de recompenza que la de ± 1, si es que la hay.

Finalmente, el algoritmo se evaluó contra un pequeño número de jugadores amateurs usando 1000 simulaciones, 4-ply search, la heurística de prioridades del tablero y la función de recompenza de ± 1. En todos estos partidos el algoritmo salió victorioso. También fue evaluado contra distintas versiones de $\alpha - \beta$ prunning, ganando UCT contra versiones que no alcanzaban nivel competitivo y perdiendo contra las profesionales. Cabe destacar que el reversi es un problema con un branching factor moderado, por lo que es esperable que $\alpha - \beta$ prunning alcance un excelente desempeño. Además de esto, UCT presenta significativas ventajas frente a $\alpha - \beta$ prunning. En primer lugar, UCT puede ser interrumpido sin haber terminado todas las simulaciones e igualmente logra un buen desempeño, ya que es equivalente a que se hubiera pasado como parámetro un menor número de simulaciones. En cambio, en esta situación, el actuar de $\alpha - \beta$ prunning es impredecible. En último lugar, el poder de $\alpha - \beta$ prunning depende casi totalmente de su función de evaluación, siendo muy débil si esta no es buena, mientras que UCT tuvo un excelente desempeño usando una heurística sencilla y ningún conocimiento extra del juego. Esto es una diferencia sustancial entre los algoritmos, ya que al fin y al cabo lo que hace UCT es aprender a jugar, mientras que $\alpha - \beta$ prunning hace una búsqueda extensiva.

5 Bibliografía

[1]: Gelly, S. and Silver, D. (2007) Combining Online and Offline Knowledge in UCT
[2]: Chaslot, G., Bakkes, S., Szita, I. and Spronck, P. (2008) Monte-Carlo Tree

Search: A New Framework for Game AI.
[3]: Rose, B. (2005) Othello: A Minute to Learn... A Lifetime to Master