UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

DESARROLLO Y ANALISIS DE UN PROGRAMA QUE JUEGA AJEDREZ

T E S I S

QUE PARA OBTENER EL TITULO DE:

M A T E M A T I C O

PRESENTA:

JOSE MANUEL MARTINEZ VILLARREAL

MEXICO, D. F. 1987

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

C O N T E N I D O

I INTRODUCCION

PRESENTACION

En esta tesis se describe y analiza un programa que juega ajedrez escrito en el lenguaje "C" para la computadora NCR Tower 1632. Este programa está dividido en tres partes fundamentales : la generación de movimientos, la búsqueda de un movimiento y la función de evaluación.

El generador de movimientos es la función encargada de generar todos los posibles movimientos de cualquier jugador desde una posición dada. Esta función depende estrechamente de la representación del tablero y las piezas dentro de la memoria de la computadora.

La búsqueda de un movimiento consiste en desarrollar y explorar un árbol de juego cuyos nodos representan posibles posiciones a las cuales se llega a través del generador de movimientos. Esta búsqueda depende de la función de evaluación, encargada de asignar un valor a cada posición a la que se llega y de esta manera poder evaluar cual es la mejor jugada.

En esta tesis se describen y comparan dos métodos diferentes de representación interna y generación de movimientos, y dos métodos diferentes de búsqueda, además de otras funciones esenciales para la operación del programa, así como algunas consideraciones sobre los problemas que surgieron al realizar el programa.

ANTECEDENTES Y ESTADO ACTUAL

La primera máquina de ajedrez que funcionó en la realidad fue construida por el científico español Leonardo Torres y Quevedo en el año 1890. Este autómata electromecánico da mate a rey negro adversario con su rey y torre propios en 63 jugadas.

Después de la invención de la computadora en los años cuarenta, empezó a revivir el interés en máquinas que jugaran ajedrez y en 1948 N. Wiener empezó a especular sobre las posibilidades de construir máquinas capaces de jugar ajedrez usando el procedimiento mini-max descubierto por Von Neumann y Morgenstern en 1944. En el trabajo pionero que realizaron sobre teoría de juegos, estos autores demostraron por primera vez que existe una solución rigurosamente matemática para todos los juegos finitos de información perfecta para dos personas tal como el ajedrez.

El 9 de marzo de 1949, Claude Shannon, investigador de los Laboratorios Bell y pionero de la teoría de la información, presentó un artículo que marca el inicio del ajedrez jugado por computadoras, aunque Shannon nunca escribió un programa que jugara ajedrez. En este artículo titulado "Programación de una computadora digital para jugar ajedrez", Shannon describe la estructura general de un programa para jugar ajedrez. Esta estructura es la que tienen hasta la fecha los mejores programas y consiste de : un árbol de juego, una función de evaluación y el procedimiento mini-max para la selección de la mejor jugada. Shannon también describió dos posibles estrategias para la selección de jugadas : A) examinar todos los posibles movimientos hasta una profundidad límite de búsqueda con la desventaja que el

programa no puede "ver" muy adelante en el juego en un tiempo razonable; B) examinar sólo los "mejores" movimientos eliminando aquellos que son malos a primera vista, lo cual permite al programa "ver" más adelante corriendo el riesgo de no examinar algún movimiento importante.

Mientras Shannon formulaba los conceptos básicos de un programa para jugar ajedrez, el matemático inglés Alan Turing también se interesó en la materia. En 1951 Turing publicó los resultados de algunos experimentos en los cuales las operaciones del programa propuesto eran simuladas a mano, este "programa" simulado resultó ser un jugador malo.

Hacia la misma época un grupo de científicos americanos del Laboratorio de Los Alamos empezó a experimentar con programas que jugaran ajedrez en la computadora MANIAC I la cual era unas 50 a 150 veces más lenta que las computadoras actuales, por lo que se hizo el programa para jugar en un tablero de 6x6, eliminando los alfiles y dos peones de cada bando. Todas las secuencias se estudiaban a profundidad de cuatro movimientos mientras que en la función de evaluación los factores principales eran el material y la movilidad de las piezas.

El primer programa que jugó ajedrez realmente fue desarrollado para correr en una computadora IBM 704, la última computadora que utilizó bulbos. Esta máquina era cerca de cuatro veces más rápida que la MANIAC I y el programa, desarrollado por un grupo de científicos encabezados por Alex Bernstein, era más complejo, usando la estrategia de búsqueda selectiva (tipo B). El programa era muy débil jugando, pero algo mejor que los anteriores.

En 1955, Alan Newell, John Shaw y Herbert Simon empezaron a trabajar en un programa para jugar ajedrez llamado CP-1, el primero que se escribió en un lenguaje de alto nivel. El uso de un lenguaje de alto nivel permitió desarrollar un programa mucho más sofisticado y complejo que cualquiera de sus antecesores, pero al usar un lenguaje de alto nivel el programa resultó ser muy lento, tanto que sólo se publicó un juego completo jugado por este programa. La mayor innovación hecha en este programa fue la introducción del procedimiento alfa-beta, el cual ha sido usado en todos los programas importantes escritos desde entonces.

El primer programa moderno fue escrito en 1966 por Richard Greenblatt en una PDP-6. Este programa, llamado MAC HACK VI utilizaba una estrategia de búsqueda selectiva (tipo B) y como el programa se escribió en un lenguaje de alto nivel, el procedimiento generador de movimientos era muy sofisticado, usando cerca de cincuenta diferentes criterios para seleccionar los mejores movimientos. En la práctica funcionó bien.

Desde este último programa hasta la fecha se han realizado muchos torneos para computadoras, de los cuales han salido muchos campeones, entre los que destacan : CHESS, al que se le adjudica una puntuación Elo de 2,200 puntos (un maestro tiene 2,300 puntos Elo y el campeón mundial unos 2,700), BELLE (2,200 puntos Elo) y el actual campeón norteamericano HITECH, escrito en el lenguaje "C" para una minicomputadora "Sun" utilizando además 64 microprocesadores de propósito especial, cada uno de ellos asignado a una de las casillas del tablero. Este último programa juega a un nivel de búsqueda promedio de 8

niveles de profundidad en el árbol, evaluando aproximadamente 175,000 posiciones por segundo con una puntuación Elo aproximada de 2,233 puntos.

ESTRUCTURA GENERAL DEL PROGRAMA Y LA TESIS

El programa está dividido en tres partes fundamentales : generación de movimientos, función de evaluación y un procedimiento de búsqueda.

El generador de movimientos tiene como función determinar todos los movimientos legales que se pueden realizar desde una posición dada. Esta función necesita de una estructura de datos que le proporcione la información necesaria tal como la posición de cada una de las piezas sobre el tablero. El generador de movimientos, así como las estructuras de datos necesarias para su funcionamiento se describen en el capítulo dos de la tesis.

El propósito de la función de evaluación es dar un valor numérico a una posición dada. Los factores para evaluar una posición son llamados heurísticas y se describen en el capítulo tres de la tesis. La mayor parte del tiempo que invierte un programa de ajedrez en obtener una jugada es consumido por esta función, ya que debe ser llamada miles de veces antes de dar una contestación.

La función de búsqueda construye un árbol de juego cuyos nodos representan posiciones posibles a las cuales se llega a través del generador de movimientos. Después se lleva a cabo una

búsqueda sobre el árbol utilizando el algoritmo mini-max o alfa-beta (se usaron ambos para comparar su eficiencia). Esta función se describe en el capítulo tres de la tesis.

En el capítulo cuatro se describen otras funciones y modificaciones que se hicieron en el programa para mejorar su eficiencia, estas son : búsquedas secundarias, aperturas de libro y selección aleatoria de movimientos.

En el capítulo cinco se habla de los posibles métodos que usarán los programas que jueguen ajedrez en el futuro para poder llegar a ganarle al campeón mundial humano. Estos métodos se están desarrollando actualmente y consisten en hacer que una máquina se asemeje a la forma en que pensamos los seres humanos al jugar ajedrez.

En el último capítulo se hace un análisis de resultados basado principalmente en la comparación entre diferentes métodos de búsqueda y representación utilizados en la elaboración del programa. Se ejemplifican algunos problemas de libro resueltos por el programa, y una partida entre el programa y una microcomputadora comercial especializada en jugar ajedrez.

II REPRESENTACION Y GENERACION DE MOVIMIENTOS

INTRODUCCION

Para que un programa juegue ajedrez, es necesario que conozca, entre otras cosas, la posición y todos los posibles movimientos de cada una de las piezas que se encuentran en un momento dado sobre el tablero. Esto se logra a través de una representación interna (en memoria) del tablero y de la posición, el color y la naturaleza de todas las piezas que están sobre él y utilizando una rutina que calcule (genere) todos los posibles movimientos válidos que puede hacer cada una de esas piezas.

Existen actualmente dos métodos para representar internamente el tablero y las piezas de ajedrez : el tradicional, sugerido por primera vez por el matemático inglés Claude Elwood Shannon y el de mapa de bits, descrito por primera vez por el investigador soviético Adel'son-Velskii. Los investigadores norteamericanos Hans Berliner, Larry Atkin y David Slate desarrollaron este último método independientemente.

En el desarrollo de esta tesis se hicieron dos programas con diferente representación interna : el primero con la representación tradicional y el segundo con mapas de bits. La intención fue comparar el tiempo de respuesta. La diferencia en los tiempos de respuesta se encuentra en el capitulo ANALISIS DE RESULTADOS.

REPRESENTACION TRADICIONAL

El tablero y la piezas

Esta representación consiste en usar una matriz de 8 x 8 elementos y designar a cada una de las piezas con un número diferente de acuerdo a su color y naturaleza, por ejemplo :

+1 = peón blanco	-1 = peón negro
+2 = caballo blanco	-2 = caballo negro
+3 = alfil blanco	-3 = alfil negro
+4 = torre blanca	-4 = torre negra
+5 = dama blanca	-5 = dama negra
+6 = rey blanco	-6 = rey negro

Cada uno de estos números se almacena en el elemento de la matriz que representa la casilla en donde está la pieza. Una casilla vacía estaría representada por un cero en el elemento de la matriz que la representa. A esta matriz de 8 x 8 elementos se le añaden cuatro renglones y dos columnas, formando así una nueva matriz de 12 x 10 elementos. A cada uno de estos elementos fuera del tablero real de 8 x 8 se le asigna un número diferente a los usados hasta ahora, por ejemplo, el 99. De esta forma es fácil detectar los bordes del tablero real. Usando esta representación, el progama puede obtener el número contenido en cualquier elemento de la matriz y así determinar el color y la naturaleza de cada una de las piezas que están sobre el tablero, además del lugar que ocupa cada una de ellas en él. Por ejemplo, si el elemento que representa la casilla P4R (e4) contiene el número +1, el progama "sabrá" que esa casilla está ocupada por un peón blanco.

Utilizando esta representación se puede hacer un esquema de la matriz y el contenido de todos sus elementos al empezar un juego :

99	99	99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99	99	99
99	-4	-2	-3	-5	-6	-3	-2	-4	99
99	-1	-1	-1	-1	-1	-1	-1	-1	99
99	0	0	0	0	0	0	0	0	99
99	0	0	0	0	0	0	0	0	99
99	0	0	0	0	0	0	0	0	99
99	0	0	0	0	0	0	0	0	99
99	1	1	1	1	1	1	1	1	99
99	4	2	3	5	6	3	2	4	99
99	99	99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99	99	99

Generación de movimientos

Es fácil generar los movimientos legales de una pieza en una posición dada si a cada uno de los elementos de la matriz anterior se le asocia un número progresivo diferente como sigue :

111	112	113	114	115	116	117	118	119	120
101	102	103	104	105	106	107	108	109	110
91	92	93	94	95	96	97	98	99	100
81	82	83	84	85	86	87	88	89	90
71	72	73	74	75	76	77	78	79	80
61	62	63	64	65	66	67	68	69	70
51	52	53	54	55	56	57	58	59	60
41	42	43	44	45	46	47	48	49	50
31	32	33	34	35	36	37	38	39	40
21	22	23	24	25	26	27	28	29	30
11	12	13	14	15	16	17	18	19	20
1	2	3	4	5	6	7	8	9	10

Así, por ejemplo, aquellos elementos de la matriz cuyo número asociado sea menor que 22 o mayor que 99 contendrán siempre el número 99, ya que están fuera del tablero real. Al iniciar un juego, el elemento con número asociado 28 contendrá el número 2 (caballo blanco), el elemento 99 contendrá el número -4, etc. De esta forma se pueden calcular los ocho números que representan los ocho elementos diferentes a los cuales se puede mover un caballo desde una posición dada añadiendo, al número que representa el cuadro donde está situado el caballo las siguientes constantes : +8, +19, +21, +12, -8, -19, -21, -12. Por ejemplo, si un caballo está en la casilla a1 (elemento número 22), los cuadros a los que se podría mover serían : 22+8=30, 22+19=41, 22+21=43, 22+12=34, 22-8=14, 22-19=3, 22-21=1 y 22-12=10. Los

posibles movimientos de un rey se calculan de la misma forma usando las siguientes constantes : +1, +9, +10, +11, -1, -9, -10, -11. El movimiento de un rey o un caballo a cualquiera de los cuadros calculados es válido siempre y cuando ese cuadro esté dentro del tablero real (no contenga el número 99), esté vacio (contenga el número 0) o contenga una pieza contraria (de signo contrario) y el rey del mismo color de la pieza que se mueve no quede en jaque después de haber realizado el movimiento. El enroque es un movimiento especial que se explicará más adelante.

La generación de los posibles movimientos para los alfiles, las torres y las damas es un poco más complicada. Para generar los posibles movimientos de un alfil es necesario seguir por separado cada una de las cuatro direcciones en las que se puede mover. Para calcular cada uno de los cuadros a los que se puede mover un alfil situado en el elemento de la matriz cuyo número es x se procede como sigue : primero se obtiene el contenido del elemento x + (11 x 1), si este cuadro está ocupado por una pieza del mismo color que el alfil o está fuera del tablero real, entonces el alfil no puede moverse a ese cuadro ni a los que le siguen en esta dirección, si el cuadro está ocupado por una pieza del color contrario, entonces el alfil puede moverse a ese cuadro pero no puede moverse a los que le siguen en esa dirección, si el cuadro está vacio, el alfil puede moverse a ese cuadro siendo necesario repetir la operación con el contenido del siguiente cuadro en esa dirección, en este caso el cuadro x+(11 x 2). Después de haber revisado el contenido de los cuadros x+(11 x 1), x+(I1x2), x+(11x3), etc., es necesario revisar los cuadros en las tres direcciones restantes : x-(11 x

1), x-(11x2), etc., x+(9 x 1), x+(9 x 2), etc. y x-(9 x 1), x-(9x2), etc. De esta forma se obtienen todos los números de los cuadros a los cuales se puede mover un alfil desde un cuadro determinado.

Para generar los posibles movimientos de una torre se procede de forma similar pero en direcciones diferentes. Suponiendo que una torre está situada en el cuadro número y, entonces se obtienen los cuadros en las siguientes direcciones : y+(10 x 1), y+(10 x 2), etc., y-(10 x 1), y-(10x2), etc., y+(1x1), y+(1 x 2), etc. y y-(1 x 1), y-(1 x 2), etc. De esta forma se obtienen todos los cuadros a los cuales se puede mover una torre desde cualquier posición determinada. Para generar los posibles movimientos de una dama, se consideran los posibles movimientos de un alfil y una torre en conjunto.

Los posibles movimientos de un peón dependen de su color por lo cual es más difícil generarlos. Supongamos que un peón blanco está situado en el cuadro número w, entonces será posible avanzar el peón un cuadro hacia adelante si el contenido del cuadro w+10 (w-10 en caso de ser negro) es cero (está vacío), de otra manera no es posible este movimiento. Si el movimiento anterior es posible y el peón está situado en la segunda línea, es posible moverlo dos cuadros hacia adelante siempre y cuando el cuadro w+20 (w-20 si el peón es negro) está vacío, de otra forma este movimiento no es posible. Para los movimientos en diagonal hacia adelante (capturas) se obtiene el contenido de los cuadros w+11 y w+9 (w-11 y w-9 si el peón es negro), si alguno de estos cuadros contiene una pieza de color contrario, entonces el

movimiento a ese cuadro es posible, de otra manera no lo es. La coronación y la captura al paso son movimientos especiales que se verán más adelante.

Una vez generado un posible movimiento es necesario saber si provoca un jaque al rey del mismo color de la pieza que se mueve, en cuyo caso el movimiento no es válido.

MAPAS DE BITS

El tablero y las piezas

Un mapa de bits consiste en un número binario de 64 bits que representa el tablero de ajedrez, cada casilla representada por un bit. Estos bits se organizan de tal manera que los primeros ocho bits representan el primer renglón, los ocho siguientes el segundo y así sucesivamente, de esta forma el primer bit (más significativo) representa el cuadro a1 y el último (menos significativo) representa el cuadro h8.

Los mapas de bits contienen información sobre las piezas o sobre cuadros del tablero de la siguiente forma : cada bit prendido puede indicar la presencia o ausencia de una pieza de cierto tipo sobre ese cuadro o indicar el cuadro al cual se puede mover alguna pieza en particular. Por ejemplo, se puede tener un mapa de bits que indique todas las casillas a las cuales se puede mover un caballo desde la casilla g3 o indicar todas las casillas que están ocupadas por un caballo blanco o todas las casillas que estén ocupadas.

Los mapas de bits básicos para describir una posición son llamados mapas de bits de localización, ya que se utilizan para obtener la localización de las piezas en el tablero de acuerdo a su color y naturaleza. Los mapas de bits de localización al empezar un juego son los siguientes :

Torres blancas

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 1
```

Caballos blancos

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 1 0
```

Alfiles blancos

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0
```

Damas blancas

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
```

Rey blanco

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
```

Peones blancos

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0
```

Torres negras

```
1 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Caballos negros

```
0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Alfiles negros

```
0 0 1 0 0 1 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Damas negras

```
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Rey negro

```
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Peones negros

```
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Utilizando estos mapas de bits se puede calcular un nuevo mapa de bits que indique todas las casillas ocupadas por piezas blancas realizando la operación lógica 'o' sobre todos los mapas de bits de localización de las piezas blancas obteniendo el siguiente resultado en la posición inicial :

Piezas blancas

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
```

El mapa de bits que representa la localización de todas las piezas negras se obtiene de la misma manera utilizando los mapas de bits de localización de las piezas negras. El mapa de bits que representa todos los cuadros que no están ocupados por una pieza blanca se calcula realizando la operación 'o exclusiva' entre el mapa de bits que tiene los 64 bits prendidos y el que representa la posición de todas las piezas blancas.

Además de los mapas de bits de localización se requieren mapas de bits de movimientos, los cuales indican los cuadros a los que se puede mover una pieza determinada desde cualquiera de los 64 cuadros. Por ejemplo, el mapa de bits que representa las casillas a las cuales se puede mover una torre desde la casilla e4 es el siguiente :

```
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
1 1 1 1 0 1 1 1
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
```

Generación de movimientos

Para generar todos los posibles movimientos de un caballo situado en la casilla x, obtenemos el mapa de bits de movimientos de un caballo desde esa casilla, el cual representa todas las casillas a las cuales se puede mover un caballo situado en ella. Una vez obtenido este mapa de bits, se realiza la operación lógica 'y' entre esta cantidad y el 'no' lógico del mapa de bits que representa la posición de todas las piezas del mismo color del caballo, obteniendo así un mapa de bits que muestra todas las casillas a las cuales es posible mover ese caballo. Utilizando el mismo algoritmo con el mapa de bits de movimientos de un rey

desde la casilla x, se pueden generar todos los posibles movimientos de un rey desde esa posición. El enroque es un movimiento especial que se verá màs adelante.

Generar los posibles movimientos de los alfiles, las torres y las damas es más complicado que generar los movimientos de cualquiera de las otras piezas. Esto se debe a que estas piezas se mueven a distancias variables en diferentes direcciones, por lo tanto es necesario calcular los cuadros ocupados en todas las direcciones sobre las cuales se puede mover la pieza de la cual se están generando sus movimientos. Esto se hace realizando la operación 'y' lógica entre el mapa de bits de movimientos de esa pieza y el 'o' lógico del mapa de bits que representa las piezas blancas y el que representa a las negras, obteniendo así un mapa de bits que contiene todos los cuadros ocupados en todas las direcciones sobre las que se puede mover la pieza de la cual se están generando sus movimientos. Una vez hecho esto, se debe seguir cada una de las direcciones desde el lugar en donde está situada la pieza hasta el borde del tablero o el primer cuadro que esté ocupado, obteniendo así todos los posibles movimientos de esa pieza. Al encontrar un cuadro ocupado es necesario saber el color de la pieza que lo ocupa, ya que de ser una pieza contraria el movimiento a ese cuadro es posible, de otra forma no lo es.

Para generar el avance de todos los peones blancos un cuadro hacia adelante, el mapa de bits de los peones blancos se corre ocho lugares hacia la derecha, resultando así un mapa de bits de todos los cuadros que están al frente de todos los peones blancos. Realizando la operación 'y' lógica entre este mapa de

bits y el mapa de bits que representa todos los cuadros vacios se obtiene un mapa de bits con todos los movimientos posibles, consistentes en avanzar un peón blanco un cuadro hacia adelante.

Para generar el avance de todos los peones blancos situados en el segundo renglón dos cuadros hacia adelante, se realiza la operación 'y' lógica entre el mapa de bits obtenido en el cálculo anterior y el que representa todos los cuadros del tercer renglón. Este resultado se recorre ocho lugares hacia la derecha y se realiza la operación 'y' lógica entre él y el mapa de bits que representa todos los cuadros vacios, obteniendo un mapa de bits que indica todos los cuadros a los cuales se puede mover un peón blanco al avanzarlo dos lugares hacia adelante desde el segundo renglón.

Para generar todas las capturas hechas por peones blancos hacia la derecha, recorremos el mapa de bits de los peones blancos nueve lugares hacia la derecha y realizamos la operación 'y' lógica entre este mapa de bits y el que representa las piezas negras, obteniendo un mapa de bits que indica todas las capturas hechas por peones blancos hacia la derecha. Para obtener las capturas hacia la izquierda se procede de la misma forma recorriendo el mapa de bits de los peones blancos siete lugares hacia la derecha.

Los posibles movimientos de los peones negros se generan de la misma manera recorriendo su mapa de bits hacia la izquierda en vez de recorrerlo a la derecha. El avance dos cuadros hacia adelante se genera utilizando el séptimo y no el segundo renglón. La captura al paso es un movimiento especial que se verá más adelante.

MOVIMIENTOS ESPECIALES

Enroque

Para poder realizar el enroque se deben cumplir las siguientes condiciones :

1.- Las casillas entre el rey y la torre están vacías y no están atacadas por piezas enemigas.

2.- El rey no está en jaque.

3.- El rey y la torre no se han movido.

Para verificar las dos primeras condiciones es necesario obtener el contenido de cada una de las casillas entre el rey y la torre del lado hacia el cual se quiere realizar el enroque y verificar que estas casillas y la casilla en donde se encuentra el rey no están atacadas. Para saber si una casilla está atacada por alguna pieza enemiga es necesario generar todos los posibles movimientos de las piezas contrarias y ver que esa casilla no es una casilla a la cual se pueda mover una pieza enemiga.

Para verificar la tercera condición, es necesario llevar tres variables : una para el rey y las otras dos para las dos torres, una variable para cada una. Estas variables son cero al principio del juego cambiando su valor a uno si el rey o la torre respectiva se mueve. De esta manera, es posible enrocarse hacia un lado siempre y cuando la variable del rey sea cero y la variable de la torre del lado al cual se quiere enrocar sea cero también, de otra manera no es posible el enroque hacia ese lado. No se habla aquí de enroque corto o largo ya que son simétricos de acuerdo al siguiente criterio : el rey, al enrocarse hacia

cualquier lado se mueve dos casillas hacia ese lado y la torre se situa en la casilla junto al rey del lado contrario del movimiento que realizó el rey.

Captura al paso

Para la captura al paso se utiliza un mapa de bits variable e independiente de todos los demás. Si algún peón, ya sea blanco o negro se mueve dos cuadros hacia adelante desde su posición inicial, entonces este mapa de bits variable contendrá un uno en la casilla que se encuentra atrás del péon una vez que se haya movido. En cualquier otro caso este mapa de bits variable es igual a cero. Después, cuando se generan las capturas de peón del contrario se realiza la operación lógica 'o' entre este mapa de bits variable y el de las piezas enemigas antes de realizar cualquier otra operación, generando así la captura al paso si es que hubiera. Se debe tener cuidado de saber cuando una captura hecha por un peón es al paso para quitar el peón capturado del tablero.

Coronación

La coronación es muy simple ya que sólo se consideró la promoción de péon por dama, por lo tanto, al llegar un peón al octavo renglón es promovido a dama y el programa sólo actualiza los mapas de bits de las damas y de los peones.

III METODOS DE BUSQUEDA Y FUNCION DE EVALUACION

INTRODUCCION

La forma natural de conocer los movimientos que pudieran ocurrir en un juego de ajedrez consiste en generar todos los movimientos legales de las piezas blancas, después todos los movimientos legales de las negras, de nuevo todos los movimientos de las blancas, etc., hasta llegar a posiciones terminales relativamente estáticas las cuales deben ser evaluadas a través de una función conocida como función de evaluación. Esta forma de conocer los movimientos futuros nos lleva a un serio problema: el número de movimientos legales desde una posición dada (en promedio 38) y la profundidad a la cual parece necesario buscar para jugar razonablemente bien (de 6 a 10 niveles o de 3 a 5 jugadas) genera un enorme número de posiciones terminales. Por ejemplo, a nivel 2 (un movimiento por jugador) el análisis de todos los movimientos legales, asumiendo que fueran 38 en cada posición, generaría 1444 posiciones terminales. Un análisis a profundidad 4 generaría 2,085,136 posiciones terminales, mientras que a profundidad 6 generaría 3,010,936,389 posiciones terminales.

A este procedimiento se le puede asociar un diagrama conocido como "árbol de juego" ya que el diagrama se asemeja a un árbol invertido, en donde la posición original es la raiz del árbol, de la cual salen las ramas principales (movimientos posibles desde esa posición) que conducen a nuevas posiciones, de las cuales se desprenden nuevas ramas secundarias (movimientos

posibles del contrario) que conducen a nuevas posiciones, etc. El punto en el cual una rama se subdivide en ramas menores es llamado "nodo" del árbol y representa una posicion intermedia en el árbol de juego.

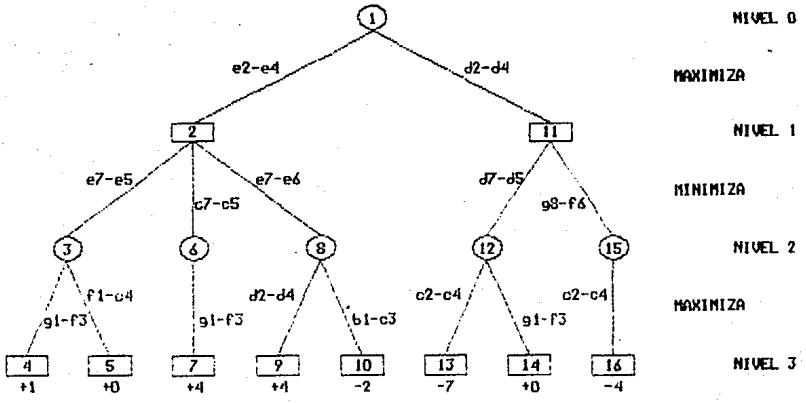
La función de evaluación se encarga de asignar valores a los nodos terminales para lo cual se basa en el material (piezas) que tiene cada jugador, la movilidad de las piezas, la estructura de los peones, etc. Esta función debe ser muy eficiente ya que es llamada una vez por cada nodo terminal. Una vez asignado un número a cada nodo terminal, es necesario "subir" estos valores hasta la raiz del árbol y de esta manera poder escoger la "mejor" jugada. Existen varios métodos para subir estos valores a la raiz, los dos más conocidos son : el algoritmo mini-max y el alfa-beta. En el desarrollo de esta tesis se hicieron dos programas, uno con el algoritmo mini-max y otro con el algoritmo alfa-beta, la intención fue mejorar y comparar el tiempo de respuesta, los resultados se encuentran en el capítulo ANALISIS DE RESULTADOS.

ALGORITMO MINI - MAX

Este algoritmo fue propuesto por primera vez por von Neuman y Morgenstern y consiste en seleccionar un movimiento haciendo una "búsqueda hacia adelante" en el árbol de juego desde el nodo raíz hasta llegar a una profundidad predeterminada, asumiendo que el jugador al cual le toca mover va a escoger la rama que representa el mejor movimiento para él. Supongamos que en la posición que representa el nodo raiz es el turno de las blancas y supongamos

además que la función de evaluación es tal que asigna valores positivos a los nodos en los cuales el blanco tiene ventaja y valores negativos si es el negro el que tiene la ventaja, entonces el procedimiento consiste en maximizar (escoger la jugada representada por la rama que lleva al nodo con el mayor valor) en los niveles de profundidad pares en donde los nodos representan posiciones en las cuales le toca mover al blanco y minimizar en los niveles de profundidad nones en donde los nodos representan posiciones en las cuales le toca mover al negro. Este proceso alternado de maximizar y minimizar es el que le da el nombre al algoritmo.

La mejor forma de explicar este algorimo es a través de un ejemplo. Supongamos que el nodo raiz representa la posición inicial del tablero al empezar un juego, como siempre, es el turno del blanco y supongamos que el programa va a analizar el árbol hasta una profundidad de tres niveles (una jugada y media) usando el generador de movimientos para desarrollar el árbol de juego siguiente :



Los círculos representan posiciones en las cuales le toca mover al blanco y los cuadros representan posiciones en las cuales es el turno del negro. Cada una de las líneas que unen a los nodos representan posibles movimientos generados por el programa. Los nodos de un árbol se pueden generar de muy diversas formas pero en los programas que constituyen el trabajo de esta tesis se usó el método "búsqueda en profundidad" que consiste en seguir una rama hasta la profundidad límite antes de seguir cualquier otra. Los nodos en el árbol anterior están numerados de acuerdo al orden en el cual los generaría el programa. Primero el programa genera los movimientos (1) e2-e4, e7-e5; (2) g1-f3, ...; evalúa la posición en el último nodo y almacena este valor, después genera los movimientos (1) e2-e4, e7-e5; (2) f1-c4, ...; aplicando la función de evaluación al último nodo y almacenando el resultado, después el programa explora los movimientos (1) e2-e4, c7-c5; (2) g1-g3, ...; evaluando y almacenando el resultado. Este procedimiento continua hasta que todos los nodos en el árbol hayan sido generados y haber evaluado los ocho nodos terminales.

Después de evaluar cada posición, el valor asignado a ese nodo es "subido" al nivel inmediato superior, siendo el mejor valor encontrado para ese nodo hasta ese momento. Cada uno de los descendientes de ese nodo es examinado de la misma manera, maximizando o minimizando según sea el caso entre el valor que se acaba de obtener y el mejor valor encontrado hasta ese momento. Si el valor asociado con la evaluación más reciente es mejor para el jugador al cual le toca mover, entonces este nuevo valor se almacena en memoria convirtiéndose en el nuevo mejor valor. Al

terminar de examinar todos los descendientes de cada nodo, el mejor valor hasta ese momento es "subido" al nivel inmediato superior. Este procedimiento termina al haber visitado y evaluado todas las posibles trayectorias del árbol.

Aunque este algoritmo puede llevar a algunas confusiones al ser aplicado por una persona, la máquina puede seguirlo fácilmente. La regla de elección en cada nodo es precisa y por lo tanto el programa sólo necesita acordarse de maximizar en niveles pares y minimizar en niveles nones.

El algorimo mini-max usado en los programas objeto de esta tesis es recursivo y se describe formalmente a continuación.

ALGORITMO MINI-MAX

```
MINI_MAX(nivel, valor, posición)
1. Si nivel = nivel_máximo entonces regresa eval(posición)
2. Si nivel es par entonces jugador = MAQUINA de otra manera
                            jugador = OPONENTE
3. lista = genera_movimientos(posición, jugador)
4. Para cada uno de los movimientos en lista :
   4.1. pos_nueva = modifica(posición, movimiento)
   4.2. Si jugador = MAQUINA
      4.2.1. valor_nuevo = MINI_MAX(nivel+1,+INF,pos_nueva)
      4.2.2. Si valor_nuevo > valor
         4.2.2.1. valor = valor_nuevo
         4.2.2.2. Si nivel = 0 entonces mejor_mov = movimiento
   4.3. Si jugador = OPONENTE
      4.3.1. valor_nuevo = MINI_MAX(nivel+1,-INF,pos_nueva)
      4.3.2. Si valor_nuevo < valor entonces valor = valor_nuevo
5. regresa(valor)
```

Descripción de las variables y constantes :

nivel : Esta variable indica el nivel al cual se está buscando en el árbol de juego.

nivel_máximo : Contiene el nivel máximo de búsqueda del algoritmo.

valor : Es el valor actual del nodo que se está examinando.

posición : Es un apuntador al tablero con la posición de las piezas del nodo que se está evaluando.

jugador : Es el jugador al cual le toca mover desde el nodo que se está examinando.

lista : Es una lista con todos los movimientos legales del jugador al cual le toca mover desde la posición indicada en el nodo que se está examinando.

movimiento : Es el movimiento que se está procesando de la lista.

pos_nueva : Es un apuntador al tablero con la posición de las piezas después de haber realizado el movimiento que se está procesando de la lista.

valor_nuevo : Es el valor que se acaba de "subir" del nivel inmediato inferior.

mejor_mov : Es una variable global a la cual se le asigna el mejor movimiento obtenido por el procedimiento hasta ese momento para ser ejecutado por la máquina.

MAQUINA : Es una constante que designa el color de las piezas con las cuales está jugando la máquina.

OPONENTE : Es una constante que designa el color de las piezas con las cuales está jugando el oponente de la máquina.

+INF : Constante considerada como +infinito (el mayor número que acepta la máquina).

-INF : Constante considerada como -infinito (el menor número que acepta la máquina).

Descripción de funciones :

eval(posición) : Llama a la función de evaluación para evaluar la posición indicada por el apuntador.

genera_movimientos(posición, jugador) : Genera todos los movimientos legales de las piezas indicadas por la variable jugador desde la posición indicada por el apuntador posición.

Al ser la función mini-max una función recursiva, es muy importante saber cuando termina la recursividad y como debe llamarse por primera vez. La forma natural de terminar la recursividad consiste en llegar al nivel máximo de búsqueda, la otra forma es cuando la lista de movimientos legales es vacía. En este último caso, regresa el peor valor posible para el jugador al cual le toca mover, ya que si no tiene ningún movimiento es necesario que esté en mate o ahogado. En los programas escritos no se consideró un rey ahogado ya que es poco común, por lo tanto, el programa detecta en cualquier momento si alguno de los reyes está en mate y considera a un rey ahogado como si estuviera en mate.

Al ser llamada por primera vez el nivel obviamente debe ser cero, el valor debe ser el peor valor que pueda obtenerse desde una posición para el jugador al cual le toca mover, en este

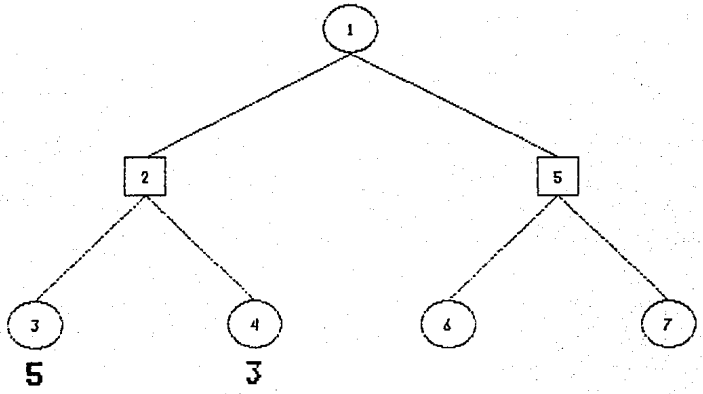
caso la máquina, por lo tanto debe ser -INF y la posición debe ser la posición actual del tablero. Por lo tanto, la primera llamada debe ser :

MINI_MAX (0, -INF, posición_actual).

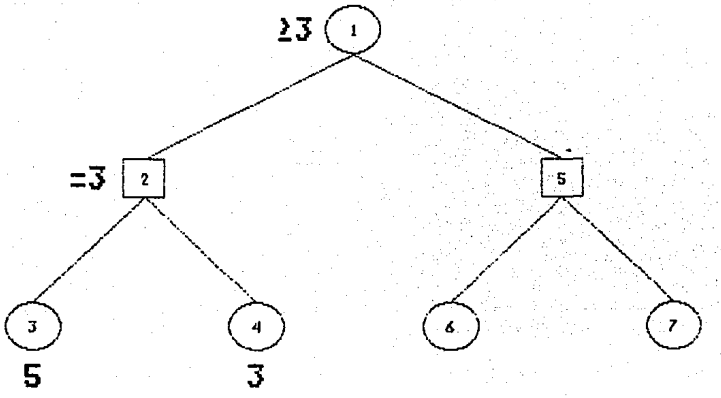
Al ejecutar esta función, la única jugada que se "sube" al nivel inmediato superior es la jugada que debe hacer la máquina en la posición en que actualmente están las piezas.

ALGORITMO ALFA - BETA

Newell, Shaw y Simon implantaron el algoritmo mini-max y al poco tiempo descubrieron que es posible ahorrar parte del trabajo que realiza el procedimiento mini-max en la búsqueda sobre el árbol de juego, esto se puede demostrar fácilmente analizando el siguiente árbol de juego :

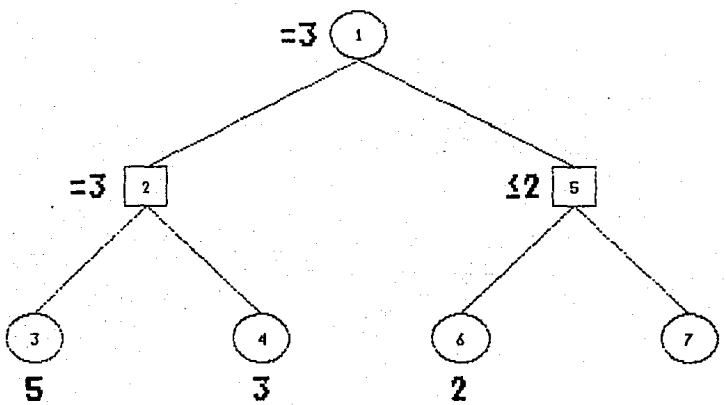


En este árbol, la función de evaluación ya asignó los valores correspondientes a los dos primeros nodos terminales. Al aplicar el procedimiento mini-max sobre estos dos nodos con los valores 5 y 3, se obtiene un valor de 3 para el nodo número 2, ya que en ese nivel se debe minimizar y por lo tanto le garantiza al jugador que minimiza un valor de 3 si el jugador que maximiza escoge la rama izquierda desde el nodo raiz, lo cual a su vez garantiza que el jugador que maximiza obtendrá un valor no menor de 3 en la raiz. Esto es claro aún antes de haber realizado cualquier otra evaluación estática de los demás nodos terminales, ya que el jugador que maximiza puede elegir la rama izquierda si la derecha resulta tener un valor menor. Esto se puede indicar en la raiz del siguiente árbol :



Ahora supongamos que se evalúa el siguiente nodo terminal y resulta tener un valor de 2, entonces el jugador que minimiza tiene garantizado un valor a lo más de 2. Usando el razonamiento

anterior, el jugador que maximiza sabe que la rama derecha desde la raíz no le puede dar un valor mayor que 2 y además, que la rama izquierda le garantiza un valor de 3, por lo tanto, es obvio que no es necesario evaluar el último nodo terminal (7), ya que no importa el valor que tenga, el nodo número cinco no podrá tener un valor mayor que 2, lo que demuestra que el valor del nodo raiz es 3, como se muestra en el siguiente diagrama :



Este procedimiento se conoce con el nombre de alfa-beta ya que el valor del movimiento que en un momento dado es el mejor para el jugador que maximiza se le designa con la letra alfa (α) y al valor del movimiento que es mejor para el jugador que minimiza se le designa con la letra beta (β). Lo interesante de este algoritmo es que el resultado final siempre es el mismo que el obtenido por el algoritmo mini-max.

A continuación se describe formalmente el algoritmo alfa-beta.

ALGORITMO ALFA-BETA

```
ALFA_BETA(nivel, posición, alfa, beta)
1.  Si nivel es par entonces jugador = MAQUINA de otra manera
                                jugador = OPONENTE
2.  Si nivel = nivel_máximo regresa eval(posición)
3. lista = genera_movimientos(posición. jugador)
4. Para cada uno de los movimientos en lista :
   4.1. pos_nueva = modifica(posición, movimiento)
   4.2. Si jugador = MAQUINA
      4.2.1. valor_nuevo = ALFA_BETA(nivel+1,pos_nueva,alfa.+INF)
      4.2.2. Si valor_nuevo › alfa
         4.2.2.1. alfa = valor_nuevo
         4.2.2.2. Si nivel = 0 entonces mejor_mov = movimiento
   4.3. Si jugador = OPONENTE
      4.3.1. valor_nuevo = ALFA_BETA(nivel+1,pos_nueva,-INF,beta)
      4.3.2. Si valor_nuevo ‹ beta entonces beta = valor_nuevo
   4.4.  Si beta › alfa entonces seguir con el siguiente
                                movimiento en lista, de
                                otra manera ir al paso 5.
5. Si jugador = MAQUINA entonces regresa(alfa) de otra manera
                                regresa(beta)
```

En este algorimo se usan las mismas variables, constantes y funciones que se usaron en el algoritmo mini_max a excepción de las variables alfa y beta, las cuales describiremos a continuación.

alfa : Representa el mejor valor encontrado en el último nivel par examinado hasta ese momento.

beta : Representa el mejor valor encontrado en el último nivel non examinado hasta ese momento.

Al igual que en el algoritmo mini-max, es necesario especificar cuando termina la recursividad y como debe llamarse por primera vez a esta función. La recursividad termina igual que en la función mini-max añadiendo además el caso en el cual beta sea menor o igual que alfa, lo cual indica que se ha encontrado el mejor valor y no es necesario buscar más. La primera llamada es semejante a la función mini-max sólo que es necesario especificar los valores de alfa y beta, que en este caso deben ser los peores valores tanto para el jugador que maximiza como para el que minimiza respectivamente. Por lo tanto, la primera llamada debe ser :

ALFA_BETA(0, posición_actual, -INF, +INF).

Como este algoritmo es más complicado que el mini-max, es apropiado ver como se aplica este algorimo a un árbol especifico, en este caso usaremos el árbol anterior de profundidad 2 con el cual explicamos el principio del algoritmo alfa-beta :

1. Al ser llamada por primera vez la función, los valores de alfa y beta son -INF y +INF respectivamente, entrando a nivel 0 (la raíz del árbol).
2. Se genera el primer movimiento (nodo número 2) y se vuelve a llamar a la función sin modificar los valores de alfa y beta.
3. Estando en el nodo número 2, se genera el primer movimiento

desde ese nodo y se vuelve a llamar la función con los mismos valores de alfa y beta, llegando al nodo número 3, que se encuentra en el nivel máximo. Por lo tanto, regresa el valor de ese nodo, en este caso 5.

4. Al subir este valor al nivel superior, se compara beta (+INF) con el nuevo valor que se acaba de subir (5), asignando el valor 5 a beta. Como beta (5) es mayor que alfa (-INF), se genera la siguiente jugada desde el nodo número 2 y se vuelve a llamar la función llegando al nodo número 4.
5. Al llegar al nodo número 4, la función regresa inmediatamente el valor de ese nodo (3) por ser el nivel máximo de búsqueda.
6. Estando de nuevo en el nodo número 2, se compara beta (5) con el nuevo valor que se acaba de subir (3), asignando este último a beta. Como ya se generaron y examinaron todo los movimientos posibles desde el nodo número 2 y al ser éste un nivel que minimiza, se regresa al nivel superior el valor de beta (3), el cual se compara con alfa (-INF), asignando 3 a alfa.
7. Estando de nuevo en el nivel 0, se genera el siguiente movimiento y se vuelve a llamar la función para llegar al nodo número 5 con los siguientes valores : alfa = 3, beta = +INF.
8. Al llegar al nodo número 5, se genera el primer movimiento y se vuelve a llamar la función, la cual regresa inmediatamente el valor del nodo número 6 (2), ya que es el nivel máximo de búsqueda.
9. Estando de nuevo en el nodo número 5, se compara el valor que

se acaba de subir (2) con el valor de beta (+INF), asignando a beta el valor 2. En este momento, se comparan alfa y beta, siendo mayor alfa lo cual termina la búsqueda en esa rama, por lo que se dice que se ha hecho un corte en el árbol de juego, regresando el valor de beta (2) al nivel superior (raíz).

10. Al estar de nuevo en el nivel 0, se compara el valor que se acaba de obtener (2) con alfa (3), conservando alfa su valor. En este momento termina la búsqueda al no haber otro movimiento en la lista, regresando el valor 3 y el movimiento asociado a él como el mejor movimiento desde la posición actual.

La eficiencia del procedimiento alfa-beta depende del orden en que se recorran las trayectorias, si las peores trayectorias se recorren primero entonces no habrá ningún corte en el árbol en cambio, si la mejor trayectoria es la primera en ser recorrida todo el resto del árbol sería cortado. Por lo tanto la eficiencia es mucho mejor si los nodos están ordenados del mejor al peor en el árbol. Knuth demostró en 1975 que si los nodos se ordenan del mejor al peor en el árbol de juego, entonces el número de nodos terminales considerados al realizar una búsqueda hasta una profundidad D utilizando el algoritmo alfa-beta es aproximadamente el mismo que el doble del número de nodos terminales generados al realizar una búsqueda hasta un nivel de profundidad D/2 utilizando el algoritmo mini-max.

FUNCION DE EVALUACION

Al escribir una función de evaluación para un programa que juega ajedrez es indispensable usar instrucciones eficientes ya que esta función es llamada por el programa miles de veces, por esta razón se hizo una función de evaluación simple la cual sólo incluye algunos factores de evaluación.

Es evidente que la evaluación debe variar a lo largo de una partida : para los finales deben regir criterios diferentes de los aplicados en las aperturas o en el medio juego. Por esta razón, en realidad son tres las funciones de evaluación, una para las aperturas, otra para el medio juego y la última para los finales, el único factor que tienen en común es el valor del material sobre el tablero en el momento de evaluar. Para este factor se consideraron los siguientes valores : peón=100, caballo=300, alfil=300, torre=500, dama=900 y rey=1000 puntos.

Para las aperturas los factores y valores que se tomaron en cuenta son los siguientes :

1. Peones doblados : Se castiga con dos puntos al jugador que tenga dos o más de sus peones en una columna.

2. Estructura de peones : Por cada peón que esté protegiendo a otro peón se da un punto. Por cada peón que esté protegido por un peón y además esté protegiendo a otro se dan seis puntos.

3. Se castiga con cuatro puntos a cualquier caballo o alfil que se encuentre en el primer renglón de su lado.

4. Se da un punto si la dama está en el primer renglón de su lado.

5. Se dan cuatro puntos al caballo que se encuentre en una casilla central ($3 \leq$ renglón ≤ 6 y $3 \leq$ columna ≤ 6).

Para el medio juego los factores con sus respectivos valores son los siguientes :

1. A cada torre que se encuentre en una columna en la cual no hay peones enemigos se le dan cuatro puntos.

2. A cada torre que se encuentre en una columna en la cual no hay peones del mismo color se le dan cinco puntos.

3. Si las dos torres se encuentran en el mismo renglón se dan dos puntos.

4. Si las dos torres se encuentran en la misma columna se dan tres puntos.

5. Se castiga con dos puntos al jugador que tenga más de un peón en una columna.

6. Por cada peón que protega a otro peón se da un punto y si además está protegido por otro peón se le dan otros siete puntos.

7. Por cada columna en la cual haya peones de un solo jugador se dan tres puntos al jugador que tiene peones en esa columna.

8. Se castiga con tres puntos a los caballos o alfiles que se encuentren en la primera línea de su lado.

En los finales los factores que se consideraron son los siguientes :

1. Se dan cinco puntos por cada torre que se encuentre en una columna en la cual no haya peones enemigos.

2. Se dan cuatro puntos por cada torre que se encuentre en una columna en la cual no haya peones del mismo color.

3. Se dan tres puntos si las dos torres se encuentran en el mismo renglón o en la misma columna.

4. Se dan dos puntos por cada torre que se encuentre en el séptimo renglon.

5. Se castiga con dos puntos por columna al jugador que tenga dos o más peones en una sola columna.

6. Si un peón está protegiendo a otro se da un punto y si además este mismo peón está protegido por otro peón se suman cuatro puntos.

7. Se dan cuatro puntos al jugador que tenga un peón en una columna en la cual el contrario no tenga peones.

Al empezar un juego se utiliza la función de evaluación de inicio de juego pasando a la de medio juego al llegar a la jugada 11. Estando en medio juego (después de la jugada 11) se pasa al final de juego cuando el número de jugadas de la máquina desde el nodo raíz sea menor que quince, el valor total de las piezas que están sobre el tablero de la máquina sea menor a 2200 o el número de piezas sobre el tablero de la máquina sea menor que nueve.

IV MODIFICACIONES ADICIONALES

INTRODUCCION

En este capitulo veremos algunas modificaciones para mejorar la calidad de juego y eficiencia del programa. Existen muchos métodos para mejorar la calidad de juego o la eficiencia de un programa que juega ajedrez, entre los cuales se encuentran : ordenamiento de los nodos del árbol de acuerdo a su valor estimado para hacer más eficiente el algoritmo alfa-beta, aperturas de libro, búsquedas secundarias, selección aleatoria de movimientos, aprovechamiento del tiempo de respuesta del contrario y funciones de evaluación sofisticadas, entre otros.

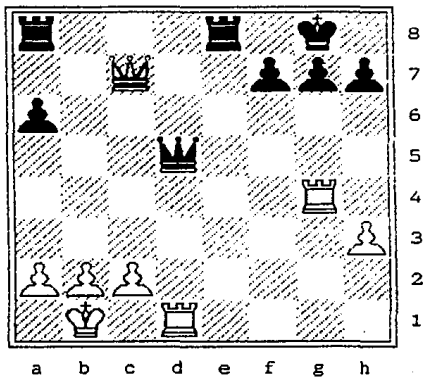
En los programas objeto de esta tesis se hicieron tres modificaciones o adiciones a los programas para mejorar la calidad de juego y la eficiencia en general del programa : búsquedas secundarias, selección aleatoria de movimientos y aperturas de libro, las cuales explicaremos con detalle a continuación.

BUSQUEDAS SECUNDARIAS

El algoritmo alfa-beta realiza una búsqueda en el árbol hasta una profundidad limite como vimos en el capitulo anterior, al llegar a esta profundidad límite la posición resultante es evaluada por la funcion de evaluación y se sube el valor asignado por ella. Sin embargo, algunas veces la posición resultante evaluada en el

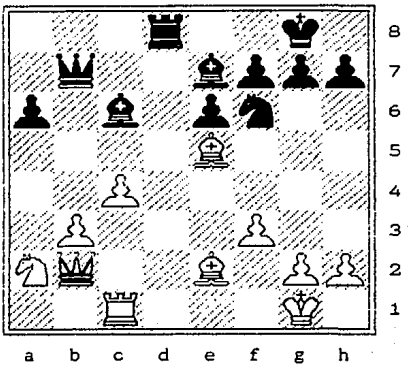
último nivel es inestable, regresando la función de evaluación un valor absurdo para esa posición. Para poder explicar mejor lo que es una posición inestable usaremos un ejemplo.

En el siguiente tablero supongamos que es el turno de la máquina la cual está jugando con las piezas blancas buscando con el algoritmo alfa-beta hasta una profundidad límite de 3 :



En esta posición una de las trayectorias de búsqueda del programa es la siguiente : (1) d1-d5, e8-e1+; (2) d5-d1, ...; al término de la cual las blancas ganan la dama negra pero quedando las blancas a un movimiento de mate, lo cual es imposible de ver por el programa ya que sólo genera el árbol hasta el nivel 3. Como el programa gana la dama negra al llegar al nivel máximo de búsqueda sin darse cuenta del peligro que corre su rey, la función de evaluación regresa un valor muy bueno para las blancas (la máquina) y el programa escoge la jugada d1-d5 sin darse cuenta que tiene perdido el juego.

Existen otras posiciones en las cuales no se pierde el juego pero si alguna pieza, esto sucede con frecuencia en los intercambios de piezas, como veremos en la siguiente posición, en la cual es el turno de la máquina con las piezas blancas.



En esta posición el programa explora la rama del árbol de juego siguiente : (1) e5-f6, e7-f6; (2) b2-f6, ...; al final de la cual las blancas ganan un caballo pero el programa no llega a ver el siguiente movimiento obvio de las negras : (2) ...,g7-f6; perdiendo las blancas la dama.

Para resolver estos problemas se realizan búsquedas secundarias, las cuales consisten en seguir la búsqueda más allá del nivel máximo de búsqueda en algunos nodos terminales considerados inestables. En el programa que se realizó con búsquedas secundarias, se tomaron en cuenta los siguientes criterios para continuar más allá del nivel máximo de búsqueda :

Se continuará la búsqueda después de llegar a un nodo del nivel máximo o de niveles más profundos siempre y cuando se cumpla alguna de las siguientes condiciones :

1. Si la jugada para llegar al nodo terminal es una captura de la pieza que el contrario movió en el nivel inmediato anterior y el valor de la pieza capturada es menor o igual al valor de la pieza que captura.

2. Si la jugada para llegar al nodo terminal es una promoción de peón por dama (coronación).

3. Si el nivel al cual se está explorando es menor o igual al nivel máximo de búsqueda más 2 y si la jugada para llegar a ese nivel no es captura pero si es jaque sobre el rey del jugador al cual le toca mover después.

4. Si el nivel al cual se está explorando es menor o igual al nivel máximo de búsqueda más 2, la jugada para llegar a ese nivel es captura y además la pieza que se está capturando es de menor valor que la pieza que captura.

5. Si el nivel al cual se está explorando es menor o igual al nivel máximo de búsqueda más 2, la jugada para llegar a ese nivel es captura, el valor de la pieza que se captura es mayor o igual que el valor de la pieza que captura y además la pieza que se captura no está protegida.

Al realizar las búsquedas secundarias es muy importante considerar que la búsqueda se puede ir a niveles muy grandes lo cual incrementaría mucho el tiempo de respuesta del programa, por

lo que es necesario tener un nivel tope, más allá del nivel máximo de búsqueda en el cual debe terminar la búsqueda no importa las condiciones del nodo terminal.

APERTURAS DE LIBRO

En ajedrez es prácticamente imposible hacer un catálogo de todas las posiciones posibles y la mejor jugada para esa posición, pero es posible hacerlo para las aperturas, las cuales están muy bien estudiadas y son fáciles de almacenar y ser seguidas por el programa, de esta manera, al principio del juego el programa pierde sólo unos segundos en encontrar la jugada en el archivo contestando rápidamente además de conseguir una buena posición al principio del juego. Esta es la razón por la cual se hizo un archivo con aperturas para ser seguidas por el programa.

Para hacer esto se utiliza un árbol de juego un poco diferente al usado por el algoritmo alfa-beta. En este árbol la raíz indica la posición inicial de un juego completo de la cual salen ramas que llevan a nodos que contienen los mejores movimientos desde esa posición. Para representar este árbol en la máquina, cada nodo tiene como información una jugada buena desde la posición que representa el nodo padre de él y dos apuntadores, uno hacia su primer hijo y el otro hacia su siguiente hermano, esto quiere decir que además de las ramas de padres a hijos tenemos ramas entre nodos al mismo nivel. De esta manera, al empezar el juego el programa ve el nodo raíz (la posición inicial), si está jugando con las piezas blancas sigue el apuntador del nodo raiz hacia su primer hijo siguiendo después

el apuntador al siguiente hermano y así sucesivamente hasta obtener todas las jugadas buenas desde la posición inicial, escogiendo alguno de los nodos hermanos aleatoriamente y realizando la jugada contenida en él. Al contestar el contrario, el programa busca la jugada que acaba de hacer el contrario entre los nodos hijos del nodo que representa la jugada que hizo el programa, de no encontrarlo sigue la búsqueda con el algoritmo alfa-beta pero si lo encuentra entonces obtiene todos los hermanos del hijo de ese nodo escogiendo uno de ellos aleatoriamente, realizando el movimiento almacenado en el nodo escogido. De esta manera el programa seguirá una apertura hasta que el árbol termine o hasta que el contrario realice un movimiento que no esté en el árbol. El algoritmo se sigue de la misma manera si el contrario es el que lleva las piezas blancas.

Para los programas objeto de esta tesis se hicieron dos programas adicionales : el primero genera el árbol de aperturas y el segundo adiciona aperturas a ese árbol. De esta manera es posible añadir aperturas al árbol en cualquier momento.

En el archivo se incluyeron sólo las aperturas más usuales hasta un nivel promedio de ocho movimientos. Las aperturas incluídas en el árbol para probar el programa son las siguientes : Ruy López, de los dos caballos, de los tres caballos, de los cuatro caballos, italiana, gambito de rey, escocesa, Giucco Piano, siciliana, inglesa y gambito de dama.

Como el programa no sabe si la jugada que está haciendo el contrario en la apertura es buena o mala y tampoco sabe si el movimiento que escogió es bueno o malo y además el árbol no contiene ninguna información adicional a este respecto, es

necesario que en el árbol de aperturas sólo se incluyan aperturas en las cuales se tiene total igualdad entre los dos jugadores tanto en material (piezas) como en posición. Lo ideal sería almacenar en cada nodo del árbol información sobre el tipo de estrategia apropiada para esa apertura lo cual no se hizo en este programa.

Una solución adoptada por gente que hace programas para jugar en torneos es seleccionar con cuidado las aperturas de libro para el tipo de juego del programa, por ejemplo, si la función de evaluación de un programa toma en cuenta factores materiales (piezas) más que factores posicionales, entonces el programador puede seleccionar aperturas de libro que rápidamente lleven a posiciones activas, en cambio si la función de evaluación toma mucho en cuenta la estructura de peones, entonces se pueden seleccionar aperturas de libro que produzcan posiciones que permitan doblar o aislar peones. Son necesarios conocimientos muy profundos del juego para poder escoger las aperturas de libro que suban el nivel de juego del programa.

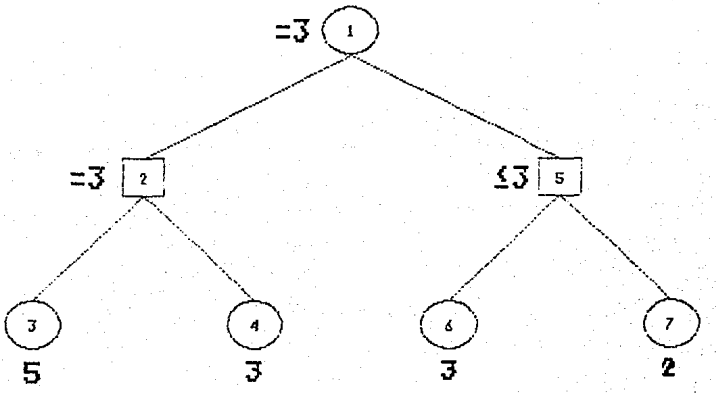
SELECCION ALEATORIA DE MOVIMIENTOS

Como las computadoras son deterministas, un programa que juega ajedrez siempre contestará con el mismo movimiento a una posición dada aún en el caso de que exista otro movimiento con el mismo valor del movimiento que se escogió, ya que siempre se genera el árbol de juego en el mismo orden y la función de evaluación regresa valores iguales para posiciones iguales. Una forma de evitar esto es hacer que el programa seleccione la jugada

aleatoriamente, es decir, si existen dos o más jugadas con el mismo valor entonces el programa seleccionará una de ellas aleatoriamente, de esta manera es posible que el programa conteste con jugadas diferentes a la misma posición.

Para hacer esto se modificó el algoritmo alfa-beta de la siguiente manera : al regresarse un valor del nivel uno al nivel cero en la variable valor, esta variable se compara con el mejor valor encontrado hasta ese momento (alfa), si el valor que se acaba de subir es igual al que ya se tenía, entonces se toma como nuevo mejor movimiento el que resulte de seleccionar aleatoriamente entre el movimiento almacenado hasta ese momento y el movimiento que se acaba de generar. De esta manera, si dos movimientos diferentes tienen el mismo valor en el nivel cero entonces se selecciona uno de ellos aleatoriamente.

Esta selección aleatoria del mejor movimiento genera un error en los cortes del algoritmo alfa-beta. La mejor forma de explicar este error es a través de un ejemplo : supongamos que el programa genera el siguiente árbol de juego :



En este árbol, el programa primero busca sobre la rama izquierda evaluando los nodos 3 y 4 obteniendo un valor de 3 para el nodo número 2, por lo tanto se le asigna al nodo raíz el valor 3 antes de seguir la búsqueda. Al explorar la rama derecha, el programa llega al nodo número 6 al cual la función de evaluación le da el valor de 3, este valor se sube al nodo número 5 el cual debe tener un valor menor o igual a 3 por ser un nivel que minimiza, pero como el valor que tiene en ese momento el nodo raíz es 3 y el valor que se acaba de subir es igual, se hace un corte en el árbol y ya no se evalúa el nodo número 7. Como el valor que se acaba de encontrar para la rama derecha (3) y el valor de la rama izquierda son iguales entonces se selecciona aleatoriamente uno de los dos lo cual es un error si se selecciona el movimiento representado por la rama derecha ya que para el nodo número 7 la función de evaluación regresaría 2 como su valor, siendo 2 el valor verdadero del nodo número 5.

Para resolver este problema es necesario hacer otra modificación al algoritmo alfa-beta. Como sólo nos interesa la selección aleatoria en el nivel cero, si un nodo del nivel uno tiene un valor igual al mejor valor encontrado hasta ese momento en el nivel cero, entonces se continúa la búsqueda sobre esa rama sin permitir cortes entre el nivel 1 y el nivel 2, de esta manera garantizamos que los dos movimientos en verdad tienen el mismo valor.

Al realizar esta modificación en el programa sólo se observó un incremento muy pequeño en el tiempo de respuesta en algunas jugadas.

V EL USO DE PLANES Y PATRONES PARA CONTROLAR LA BUSQUEDA

INTRODUCCION

Los mejores programas que juegan ajedrez actualmente no razonan de la misma manera que lo hacen los grandes maestros. Los grandes maestros, cuyo juego es todavía mucho mejor que el de cualquier programa, parecen tener un gran número de "patrones" almacenados en la memoria y, al analizar una posición, utilizan alguno de estos patrones para sugerir un plan de ataque o defensa. Posteriormente este análisis es verificado o corregido al realizar una pequeña búsqueda sobre el árbol de juego.

Los programas actuales utilizan muy pocos conocimientos del juego de ajedrez, generalmente contenidos en la función que genera movimientos y en la función de evaluación. Estos programas no contienen en ningún lugar el tipo de conocimientos basados en patrones característico de los seres humanos, por lo cual no pueden desarrollar ideas conceptuales sobre una posición determinada, aun así jugando bastante bien. Una de las cosas más interesantes que han demostrado los programas que juegan ajedrez es que la búsqueda a gran escala puede encubrir muy bien la ignorancia.

Uno de los principales problemas que tienen los programas basados en una búsqueda hasta una profundidad límite en el árbol, como el escrito para esta tesis, es que no pueden encontrar combinaciones más allá de ese límite a menos que sea encontrada a través de una búsqueda secundaria. Una manera tentativa de

resolver este problema es hacer que el programa utilice una gran cantidad de conocimientos sobre ajedrez para controlar la búsqueda y de esta manera poder llegar a una profundidad arbitraria en el árbol.

En este capítulo se analiza la forma en que piensa un gran maestro al jugar ajedrez y cuales serian los elementos necesarios para poder hacer un programa que jugara ajedrez en forma semejante a como lo hacen los grandes maestros. Es necesario aclarar que para este trabajo de tesis no se hizo ningún programa con estas características.

COMO PIENSA UN GRAN MAESTRO DE AJEDREZ

La diferencia más importante que se ha observado entre un gran maestro de ajedrez y un jugador promedio consiste en la percepción inicial de la posición, ya que un gran maestro descubre importantes jugadas y características en una posición en cuanto la ve. Algunos experimentos han revelado que esto se debe a una habilidad especifica para el ajedrez y no a una capacidad de memoria fuera de lo común. Los grandes maestros de ajedrez tienen almacenados en su memoria una gran cantidad de patrones de ajedrez desarrollados a lo largo del tiempo, los cuales almacenan junto con información especifica acerca de ellos, lo que les permite descubrir información importante de una posición cualquiera debido a su semejanza con algunos patrones ya almacenados. Se estima que un gran maestro tiene almacenados en

su memoria entre 10,000 y 100,000 patrones diferentes, cada uno de los cuales consiste en el patrón en sí e información importante acerca de él.

Al parecer, los grandes maestros hacen un análisis basado en los patrones almacenados en su memoria, lo que les permite sugerir jugadas y planes, pero aún así deben hacer una pequeña búsqueda para verificar ese análisis. La gran mayoría de las ideas actuales acerca de como piensa un jugador de ajedrez se deben a algunos experimentos en los cuales un gran maestro piensa en voz alta al resolver un problema de ajedrez. Ellos buscan hacia adelante, realizando mentalmente las mejores jugadas de cada lado para investigar el resultado de un movimiento en particular. A esta búsqueda se le ha dado el nombre de "búsqueda progresiva", debido a que en muchas ocasiones regresan a buscar sobre una jugada que ya analizaron, sólo que a una profundidad mayor. Esta segunda búsqueda se inicia al descubrir alguna información que puede servir para esta jugada al estar analizando otra. Los grandes maestros utilizan un conocimiento heurísitico (supuestamente obtenido de sus patrones almacenados) para restringir la búsqueda a unos cuantos movimientos, desarrollando por lo general menos de 50 nodos del árbol de juego terminando la búsqueda sobre una línea de juego al realizar algún tipo de evaluación aparentemente basada en los patrones ya almacenados en su memoria.

Además de las habilidades anteriores, los seres humanos pueden modificar fácil y rápidamente sus conocimientos almacenados en forma de patrones, esto es evidente ya que los grandes maestros no suelen cometer el mismo error dos veces.

Resumiendo, los grandes maestros utilizan sus patrones almacenados en su memoria para analizar una posición y obtener un plan de ataque o defensa con el cual se controla una búsqueda que verifica o corrige el resultado del análisis inicial.

PLANES Y PATRONES

El método más fiel para escoger la mejor jugada en un juego de ajedrez consiste en conocer todas las posiciones que pueden ocurrir en un juego así como la mejor jugada para cada una de ellas, esto es imposible de llevar a cabo debido al número gigantesco de las posiciones que pueden ocurrir en un juego, pero sí es una solución para las aperturas, para las cuales se puede tener un catálogo con las posiciones y las mejores jugadas desde cada una de ellas, pudiendo hacer estas jugadas correctamente en muy poco tiempo.

En cualquier posición del medio juego es posible encontrar patrones familiares como son : peones doblados, enroque, tenedores de caballos, piezas alineadas, etc., así como ciertos aspectos estructurales de la formación de peones y otros aspectos que pueden ser reconocidos como patrones inmediatamente, los cuales no están asociados con movimientos especificos pero sí con algunas metas que el jugador puede intentar de las cuales se pueden desarrollar varios planes para conseguirlas

Al igual que en el medio juego, en los finales se deben utilizar patrones que normalmente estarán asociados con algunas metas especificas, de los cuales se pueden desarrollar planes en lugar de almacenar movimientos especificos.

Alexander Kotov, un gran maestro soviético, escribió en su libro "Think like a Grandmaster" : "...es mejor seguir un plan, aun no siendo el mejor, que jugar sin ninguno." Los programas actuales hacen con frecuencia dos jugadas consecutivas que no tienen ninguna relación entre ellas, esto sucede con mayor frecuencia al final de un juego. También les cuesta trabajo a los jugadores novatos desarrollar planes, ya que después de controlar el centro, desarrollar sus piezas y haberse enrocado no saben como continuar y, por lo tanto, hacen movimientos sin relación alguna o simplemente esperan una oportunidad para realizar un ataque.

En un programa basado en planes y patrones, el tiempo que se invierte en analizar una posición se debe recuperar al buscar en un árbol de juego muy pequeño. Utilizando los patrones se analiza una posición y se obtiene una meta, con lo cual se puede hacer un plan, y con él, controlar la búsqueda.

Los pasos a seguir en un programa con estas características serían los siguientes : primero se hace un análisis estático de la posición actual en base a los conocimientos almacenados (patrones), con lo cual se puede obtener una meta a realizar generando un plan para llegar a ella, entonces se generan todos los movimientos posibles y se intenta obtener uno de ellos, el más consistente con el plan obtenido con anterioridad.

Se pueden obtener varios planes para intentar llegar a una meta en particular, pero si no hay ningún movimiento del que resulte una evaluación positiva con respecto a la meta original, entonces debe escogerse otra meta y repetir el procedimiento. Si

no es posible conseguir otra meta entonces se toma el primer movimiento que no lleve a una posición con desventaja después de haber buscado en el árbol hasta cierta profundidad.

No es necesario repetir el análisis estático realizado al inicio de la búsqueda ya que los planes obligarían al programa a concentrarse sólo en ciertas jugadas reduciendo considerablemente el factor de ramificación del árbol al darle una dirección a la búsqueda, controlándola.

ESTADO ACTUAL

Hasta el dia de hoy son pocos los programas que se han escrito intentando realizar una exploración del árbol basada en un número mayor de conocimientos de ajedrez. De estos programas los más importantes e interesantes son : el programa escrito por Jacques Pitrat y el escrito por David Wilkins llamado PARADISE. Ninguno de estos dos programas juega una partida completa de ajedrez, sólo resuelven problemas interesantes de medio juego.

El programa de Pitrat fue escrito en el lenguaje FORTRAN en 1976. A este programa es necesario indicarle cuanto material (piezas) debe ganar para que pueda resolver un problema. El programa analiza la posición inicial usando una base de conocimientos la cual sólo consiste en ciertos tipos de ataques y con ella poder desarrollar un plan. El plan generado es utilizado para controlar la búsqueda, nunca volviendo a realizar un análisis tan detallado como el que se hizo en la posición inicial, aunque realiza un análisis muy limitado para modificar el plan ocasionalmente durante la búsqueda, sólo permitiendo dos

cambios en el plan original. Este programa sólo resuelve un pequeño número de problemas sin tener un límite de profundidad de búsqueda del árbol, pudiendo encontrar combinaciones a una profundidad de 20 o más niveles, generando miles de nodos al resolver problemas difíciles.

PARADISE, el programa escrito en el lenguaje LISP por David Wilkins en 1979 también necesita información sobre la cantidad de material (piezas) que debe ganar para poder resolver el problema. El programa analiza la posición inicial usando su base de conocimientos, consistente en cerca de 200 reglas de producción, cada una de las cuales tiene un patrón como condición. Todas las producciones que tengan un patrón similar al de la posición propuesta transmiten un concepto en una base de datos para ser usados posteriormente por el sistema en el proceso de razonamiento. Cada concepto consiste en un nombre, unos valores iniciales para ciertas variables y una lista de razones por las cuales se ha escogido ese concepto. La base de datos es un pizarrón global en donde las producciones escriben información, utilizándola posteriormente para llevar a cabo la búsqueda sobre el árbol de juego y de esta manera demostrar que el movimiento sugerido por el análisis basado en patrones es en verdad el mejor. Este programa resuelve un número más grande de posiciones que el de Pitrat, encontrando combinaciones a una profundidad de 20 o más niveles visitando a lo más 500 nodos del árbol de juego.

VI ANALISIS DE RESULTADOS

REPRESENTACION TRADICIONAL VS. MAPAS DE BITS

En esta sección se comparan los dos programas hechos con representaciones diferentes, el primero de ellos utilizando la representación tradicional sugerida por primera vez por Shannon y el segundo con la representación basada en mapas de bits como ya se expuso en el capítulo dos. Ambos programas utilizan el algoritmo alfa-beta para realizar la búsqueda utilizando funciones de evaluación iguales, ninguno de los dos programas usa aperturas de libro ni búsquedas secundarias.

Para realizar la comparación se jugaron tres juegos completos con cada uno de los programas, realizando los programas una búsqueda con límite de profundidad de dos niveles, tomándose el tiempo de respuesta en cada jugada y calculando el promedio de los tres juegos. Los resultados son los siguientes :

El tiempo de respuesta promedio del programa con representación tradicional fue de 19.2 segundos para cada jugada, mientras que el tiempo promedio de respuesta del programa usando una representación basada en mapas de bits fue de 7.1 segundos por jugada, lo cual indica que se consiguió reducir a menos de la mitad el tiempo de respuesta al utilizar mapas de bits para representar el tablero y las piezas.

MINI-MAX VS. ALFA-BETA

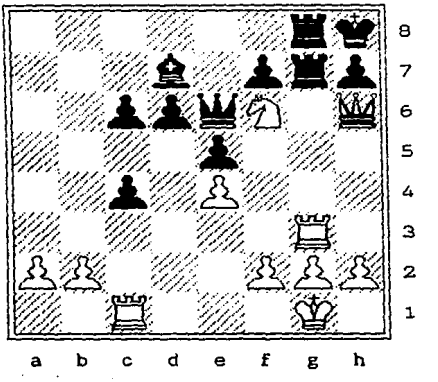
Para comparar la eficiencia entre el algoritmo mini-max y el algoritmo alfa-beta se hicieron dos programas que sólo difieren en el algorimo de búsqueda, uno con el algoritmo mini-max y el otro con el alfa-beta. Ambos programas utilizan aperturas de libro y representación a base de mapas de bits pero ninguno de ellos realiza búsquedas secundarias.

Para llevar a cabo la comparación se tomaron 10 posiciones, las cuales resolvieron los dos programas tanto a nivel dos como a nivel tres. Los factores que se tomaron en cuenta son : el tiempo de respuesta en segundos, el número de nodos totales generados en la búsqueda y el número de nodos terminales evaluados, tomando el promedio que tomó cada programa en cada posición para cada uno de estos factores. Los resultados son los siguientes :

	N I V E L 2			N I V E L 3		
	TIEMPO (SEG.)	NODOS TOTALES	NODOS EVALUADOS	TIEMPO (SEG.)	NODOS TOTALES	NODOS EVALUADOS
MINI-MAX	17.2	1,221	1,192	269.4	26,652	25,458
ALFA-BETA	8.7	875	781	48.3	4,446	4,196

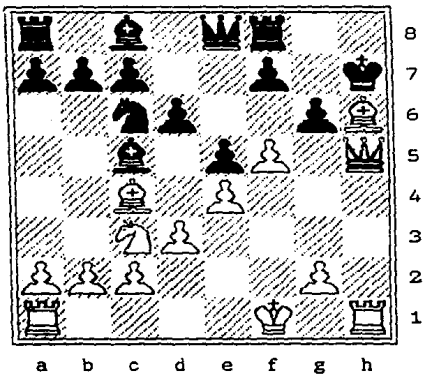
PROBLEMAS DE LIBRO

En el siguiente diagrama se muestra una posición en la que es el turno de las blancas, las cuales pueden dar mate en dos jugadas de la siguiente manera : 1. h6-h7+, g7-h7; 2. g3-g8, mate.



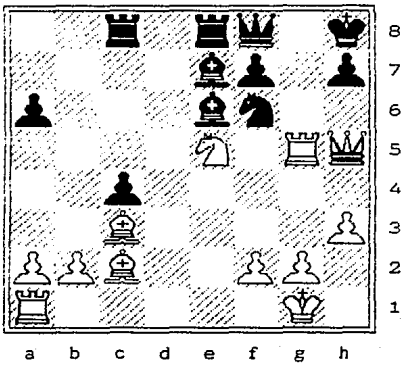
Este problema lo resolvió el programa buscando a una profundidad limite de tres niveles, generando en total 12,301 nodos de los cuales 6,118 fueron nodos terminales evaluados, llegando a un nivel de profundidad de siete niveles utilizando búsquedas secundarias, lo cual le tomó un tiempo de 4 minutos y 58 segundos, realizando el movimiento esperado : h6-h7.

En la siguiente posición las blancas dan mate en dos jugadas de la siguiente manera : 1. h5-g6+!, f7-g6; 2. h6-f8, mate.



Este problema lo resolvió el programa jugando 1. h5-g6 tomando un tiempo de 1 minuto y 13 segundos, realizando una búsqueda total hasta el nivel dos y búsquedas secundarias hasta el nivel seis del árbol, generando en total 2,733 nodos de los cuales 1,588 fueron nodos terminales evaluados.

En la siguiente posición también deben dar mate las blancas en dos movimientos, la solución es la siguiente : 1. h5-h7+!, f6-h7; 2. e5-f7, mate. En esta posición también es posible otro mate con una jugada más : 2. e5-g6+, h8-g8; 3. g6-e7, mate.



El programa contestó en 6 minutos y 53 segundos el movimiento esperado : h5-h7, buscando totalmente en el árbol hasta el nivel tres y realizando búsquedas secundarias hasta el nivel 12, generando en total 16,975 nodos, de los cuales 8,406 fueron nodos terminales evaluados.

PARTIDA

Blancas : Programa, utilizando mapas de bits, aperturas de libro, el algoritmo alfa-beta y búsquedas secundarias a un nivel máximo de búsqueda total de dos niveles.

Negras : "Chess challenger sensory voice", microcomputadora comercial especializada para jugar ajedrez, también a nivel de juego de dos niveles.

	Blancas	Negras
1.	e2-e4	e7-e5
2.	f1-c4	g8-f6
3.	d2-d3	c7-c6

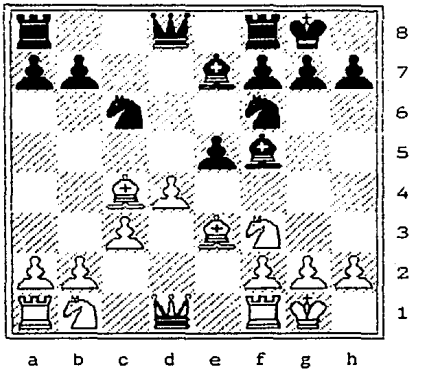
En este momento terminan los dos programas las aperturas de libro.

4.	c1-e3	d7-d5
5.	e4-d5	f8-b4+

Las negras pierden un peón al realizar esta última jugada, pero el programa "Chess challenger" lo hace seguramente porque le da muchos puntos el dar un jaque.

6.	c2-c3	b4-e7
7.	d5-c6	b8-c6
8.	g1-f3	e8-g8 (O-O)
9.	e1-g1 (O-O)	c8-f5
10.	d3-d4	. . .

En este momento el programa con las blancas termina la apertura para empezar con el juego medio. La posición después de este movimiento es la siguiente :



10.	. . .	e5-e4
11.	f3-e5	c6-e5
12.	d4-e5	d8-d1
13.	f1-d1	f6-g4
14.	e3-d4	f8-d8
15.	b1-a3	e7-a3

Este último cambio lo llevan a cabo las negras ya que consiguen doblar peones blancos, lo cual le da puntos.

16.	b2-a3	a8-c8
17.	c4-b5	a7-a6
18.	b5-e2	d8-d5
19.	e2-g4	f5-g4
20.	d1-d2	c8-d8
21.	a1-b1	b7-b5
22.	d2-b2	. . .

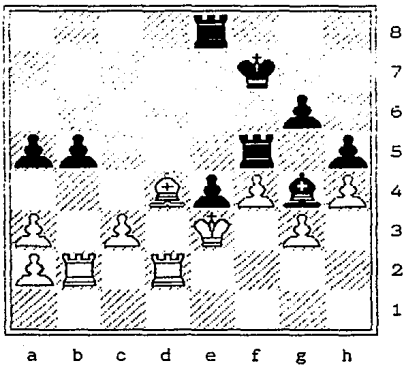
A falta de iniciativa, las blancas doblan sus torres consiguiendo algunos puntos.

22.	. . .	g4-f5

En este momento ninguno de los dos programas tiene algún plan para atacar, por lo cual sólo hacen jugadas que les den más puntos mejorando un poco su posición.

23.	g1-f1	h7-h5
24.	h2-h4	d8-d7
25.	f1-e2	g8-f8
26.	g2-g3	g7-g6
27.	e2-e3	d7-d8
28.	b2-d2	f8-g7
29.	b1-d1	a6-a5
30.	f2-f4	d8-c8
31.	e5-e6+	g7-g8
32.	e6-f7	g8-f7
33.	e3-f2	f4-g5
34.	d1-b1	c8-e8
35.	b1-b2	d5-f5
36.	f2-e3	f5-d5
37.	e3-f2	d5-f5
38.	f2-e3	f5-d5

Terminando tablas por repetición al no poder, ninguno de los dos programas, hacer más puntos para reforzar su posición y ser incapaces de desarrollar un plan. La posición al finalizar el juego es la siguiente :


8
7
6
5
4
3
2
1
a b c d e f g h

VII CONCLUSIONES

Los programas que se hicieron para esta tesis están basados en una búsqueda total, es decir, consideran todos los posibles movimientos desde una posición dada para escoger un movimiento. Lo que se hizo fue mejorar el tiempo de respuesta de varias maneras para conseguir que el programa llegara a un nivel más profundo dentro del árbol de juego y de esta manera jugara mejor. Se consiguió mejorar el tiempo de respuesta a través de una representación más apropiada para una computadora, del algoritmo alfa-beta y de las aperturas de libro. Todavía es posible mejorar relativamente el programa con respecto al tiempo de respuesta utilizando una computadora más rápida, tratando de utilizar el tiempo de respuesta del contrario y mejorando en general el programa para hacerlo más eficiente.

En cuanto a la calidad de juego, se intentó hacer una función de evaluación sofisticada, pero esto sólo aumentó el tiempo de respuesta en forma dramática por lo que se optó por hacer las búsquedas secundarias, lo cual mejoró considerablemente la calidad de juego del programa, realizando una búsqueda parcial en el árbol de juego más allá del nivel máximo de búsqueda total.

Las heurísticas utilizadas para realizar las búsquedas secundarias pueden mejorarse o ampliarse para mejorar tanto el control de la búsqueda como la calidad de juego del programa. Sin embargo, para lograr mejorar substancialmente la calidad de

juego del programa, habría que incluir algunas de las consideraciones mencionadas en el capítulo cinco acerca de planes y patrones.

Desde el tiempo en que Shannon escribió su artículo pionero en 1949 se han logrado resultados sorprendentes, como es el caso del programa campeón norteamericano HITECH, el cual está jugando al nivel de un maestro internacional (2,233 puntos Elo), por lo que es muy probable que dentro de poco tiempo (20 años o menos) exista un programa que juegue al nivel del ser humano campeón del mundo, aunque va a ser necesario que este programa utilice algunos de los métodos mencionados en el capítulo sobre planes y patrones o algún otro método nuevo, ya que si sólo utiliza el método de la fuerza bruta deberá correr en una computadora por lo menos 3,500 millones de veces más rápida que las computadoras más rápidas existentes hasta este momento.

REFERENCIAS Y BIBLIOGRAFIA

Berliner, Hans J., "Some necessary conditions for a master chess program", Third International Joint Conference on Artificial Intelligence, Stanford, CA, 1973.

Berliner, Hans J., "Chess as problem solving : The Development of a tactics analyzer", tesis doctoral inédita, Carnegie-Mellon University, Pittsburgh, PA, 1975.

Clarke, M. R. B., "Advances in computer chess 3", Pergamon Press, Oxford, 1982.

Cracraft, Stuart M., "Bitmap move generation in chess", ICCA Journal, septiembre 1984, 146-153.

De Groot, A. D., "Thought and choice in chess", The Haugue : Mouton, 1965.

Frey, Peter W. (editor), "Chess skill in man and machine", segunda edición, Springer-Verlag, Nueva York, 1983.

Gillogly, James J., "The Technology chess program", Artificial Intelligence, vol. 3, núm. 3, 1972.

Knuth, D. E. y Moore, R., "An analysis of alpha-beta pruning", Artificial Intelligence, vol. 6, 1975, 293-326.

Kotov, A., "Think like a Grandmaster", Chess Digest, Dallas, 1971.

Pachman, L. y Kühnmund, Vas I., "Ajedrez y computadoras", Martinez Roca, Barcelona, 1982.

Pitrat, Jacques, "A Chess combination program which uses plans", Artificial Intelligence, vol. 8, núm. 3, 1977.

Pitrat, Jacques, "A program for learning to play chess", en Chen, C. H. (editor), "Pattern recognition and Artificial Intelligence", Academic Press, Nueva York, 1976.

Rich, Elaine, "Artificial Intelligence", McGraw-Hill, Nueva York, 1983.

Shannon, Claude E., "Programming a Digital Computer for Playing Chess", Philosophy Magazine, vol. 41, 1950.

Welsh, David E., "Computer Chess", Wm. C. Brown Publishers, Dubuque, Iowa, 1984.

Wilkins, D. E., "Ussing patterns and plans to solve problems and control search", AIM-329, Computer Science Department, Stanford University, 1979.

Wilkins, D. E., "Ussing patterns and plans in chess", Artificial Intelligence, vol. 14, 1980, 165-203.

Wilkins, D. E., "Using knowledge to control tree searching", Artificial Intelligence, vol. 18, 1982, 1-51.

Winston, Patrick H., "Artificial Intelligence", segunda edición, Addison-Wesley, Reading, Mass., 1984.