Solving Connect-4 on medium boardsizes

John Tromp[1]

CWI, Amsterdam, The Netherlands

ABSTRACT

We present game theoretical values of connect-4 on board sizes up to width+height=15, highlighting some features of the publicly available program, Fhourstones (Tromp, 1995), that produced the results.

1. INTRODUCTION

Connect-4 is one of the first non-trivial games to be solved by computer. In October 1988, James Allen and Victor Allis (Allis, 1988), independently, announced their proofs that the game is a first player win. In 1995, the author published a database mapping each 8-ply position (that has no winning or forced move) to its minimax value. Since solving positions deeper than 8-ply takes mere seconds with a program such as Fhourstones, this database can be considered to *strongly* solve the game as played on the standard 7x6 board.

2. FHOURSTONES

The name is a pun on the well-known Dhrystone integer benchmark—with dhry sounding just like the German word for three (drei), Fhourstones can be considered a Dhrystone++. The program uses exhaustive alpha-beta search with the following features.

2.1 Board Representation

The board is represented with two bitboards, B_1, B_2, one for each player. We number the columns from 0 through width $-$ 1 and the rows from 0 (bottom) through height $-$ 1. The bitboard is of size width(height $+$ 1) and stores the cell at column x and row y in bit x(height $+$ 1) $+ y$. An 8x7 board is the largest whose bitboards fit in 64 bits, one reason why we limited ourselves to width+height=15.

6	13	20	27	34	41	48
5	12	19	26	33	40	47
4	11	18	25	32	39	46
3	10	17	24	31	38	45
2	9	16	23	30	37	44
1	8	15	22	29	36	43
0	7	14	21	28	35	42

Table 1: Bitboard layout

Table 1 shows this layout for the standard board size. The extra row of 0 bits on top allows for very efficient implementation of two crucial operations, namely testing for a win, and computing hash-values.

[1]email:john.tromp@gmail.com

2.2 Testing for a win

The Java function

```
boolean haswon(long bitboard) {
  long y = bitboard & (bitboard>>HEIGHT);
  if ((y & (y >> 2*HEIGHT)) != 0) // check diagonal \
    return true;
  y = bitboard & (bitboard>>(HEIGHT+1));
  if ((y & (y >> 2*(HEIGHT+1))) != 0) // check horizontal -
    return true;
  y = bitboard & (bitboard>>(HEIGHT+2)); // check diagonal /
  if ((y & (y >> 2*(HEIGHT+2))) != 0)
    return true;
  y = bitboard & (bitboard>>1); // check vertical |
  return (y & (y >> 2)) != 0;
}
```

uses 8 shifts, 8 and's, and 4 comparisons to determine if a player's bitboard contains 4 in a row. The extra row of 0 bits on top prevents wrap-around diagonals like $(8, 14, 20, 26)$ on 7×6 from generating a false win.

2.3 Computing hash values

Define the bitboard BOTTOM to have all the row 0 (bottom) bits set. If we take the sum $B_1 + B_2$, which is a bitboard of all stones, and add BOTTOM to it, then in each column we are adding in binary $11..1 + 1 = 100...0$ (now the extra row on top is needed for full columns). So $B_1 + B_2 + BOTTOM$ has a unique 1-bit in each column, indicating its height. Let's call this value the "skyline" of the position. We can thus fully describe the board by adding the skyline to B_1. This position code, which equals $2B_1 + B_2 + BOTTOM$, uniquely determines both the skyline and B_1, and hence also B_2.

2.4 The transposition table

The position code, of size width(height $+ 1$), is used as a hash value in the transposition table, which takes advantage of the huge number of transpositions in connect-4. Dividing the hash value by the table size (a prime number) gives a remainder, called the hash index, which is used to select an entry in the table. Our table uses the "TwoBig" replacement scheme of (Breuker, 1998a; Breuker, 1998b), in which every table entry has two slots, *big* and *new*. Each slot has a lock field and a 3 bit score field. The lock field holds the least significant l bits of the hash value, where l is the locksize. The score is one of $\{-, <, =, >, +\}$ representing a loss, a loss-or-draw, a draw, a draw-or-win, and a win, respectively. In addition, a 6 bit field called "work" represents the amount of effort, on a logarithmic scale, that went into computing the big score. When a new entry is to be written at this index, it overwrites the new slot, unless the work performed is at least the one stored (or the lock matches the big one). Thus, the big slot always holds the most expensive result with this hash index, while the new slot holds the most recent one.

2.5 Minimum transposition table size

Since the table size is chosen to be some prime P, positions that match both in index and in lock must be congruent modulo $2^l P$. Thus $\log_2 P >$ width(height $+ 1$) $- l$ suffices to avoid collisions. We can actually relax this by a few percent ($1/32$ to be precise), since the 5 most significant bits of the position code cannot be all 1, unless player 1 already has won in the top 4 squares of the last column. For smaller boardsizes like 7×6 the choice of $l = 26$ gives a convenient hash entry size of $2(26 + 3) + 6 = 64$ bits, and doesn't require a very big table (8M entries). Bigger boards like 8×7 will require a larger locksize, like $l = 36$, to keep table size within reason.

2.6 Move ordering

Moves are dynamically ordered by the history heuristic. For each side, and each square, an integer history$[side][i]$ represents the current desirability for that side to occupy that square. It is initially set to the number of possible winning configurations (length 4 line segments) containing the square, creating a bias toward occupying central squares. Whenever a beta cutoff occurs, each earlier move tried is debited 1, and the total (possibly 0) is then credited to the move producing the cutoff, thus preserving the sum history of all squares.

3. RESULTS

We ran Fhourstones on all boardsizes where width+height is at most 15, the results of which are shown in Table 2. With an 8306069 entry transposition table, the standard size 7×6 is solved in less than 5 minutes, searching 1479113766 positions. The most time consuming size is 9×6, requiring all 5 unique first moves to be tried to realize the loss, searching $\sim 2 \cdot 10^{13}$ positions in about 2000 hours on a 1.4Ghz Opteron 840, with a 268435399 entry transposition table size.

11	=							
10	=	=						
9	=	=	+					
8	=	=	-	+				
7	=	=	+	=	+			
6	=	=	-	+	-	-		
5	=	=	=	=	+	+	+	
4	=	=	-	=	-	-	-	-
	4	5	6	7	8	9	10	11

Table 2: Connect-4 values on medium boardsizes.

4. FUTURE DIRECTIONS

Allis showed how $6 \times (2n)$ is at least a draw for player 2. Our results show that she can do even better for $n = 2, 3, 4$. Formulating a simple winning strategy for these games could lead to a proof that $6 \times (2n)$ is a 2nd player win for all n. Toward this end, we ran some experiments by changing the search routines. Forcing the 2nd player to answer any 1st player move in columns B or E in the same column, changed the wins into draws. So the 2nd player cannot simply claim all even squares in these columns. Still there are overwhelming similarities in the winning strategies for these three heights. We encourage the reader to download the code and try their own experiments.

The next diagonal width+height=16 contains two interesting instances. The 6×10 case will provide either more support or a disproof of the $6 \times (2n)$ conjecture, while 8 is a popular size on internet game servers. Both of these require modification of Fhourstones to support bitboards larger than 64bits, perhaps using gcc's __uint128_t datatype.

5. REFERENCES

Allis, V. (1988). A Knowledge-based Approach of Connect-Four.

Breuker, D. (1998a). *Memory versus search in games.* Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands.

Breuker, D. (1998b). Ph.D. thesis: Memory versus Search in Games. http://www.xs4all.nl/ breukerd/thesis/.

Tromp, J. (1995). John's Connect Four Playground. http://www.cwi.nl/ tromp/c4/c4.html.