Attack and Defend Maps (/Attack+and+Defend+Maps)

Samuel Bak - Sheltering Myths, 1998 [1]

Attack and Defend Maps,
also called **Attack Tables**, refer to data-structures, most often arrays, containing attack or defend information for every pawn or piece and/or the transposed information for each square, which pieces control, that is either attack or defend it. These Maps are useful for evaluation purposes such as safe mobility, SEE and of course move generation. While the piece centric attack information, a set of attacked squares per piece, is often encoded as bitboard, there are more alternatives for storing the square centric information, about attacking pieces.

Table of Contents

- Maintaining Attacks
  - Incremental Update
  - On the Fly
- Implementations
  - Classical Approach
  - Alternatives
    - Piece-Sets
    - Ed's lookup
    - Direction wise
- See also
- Forums Posts
  - 1995 ...
  - 2000 ...
  - 2005 ...
  - 2010 ...
- References
- What links here?

Maintaining Attacks

Incremental Update

The piece centric and/or square centric information is often initialized at the root and updated incrementally during the search while making and unmaking moves. The idea is that a move has often only a local influence on the attack tables, and that it is usually cheaper to change only those squares which changed from- or to-attacks, rather than all squares from scratch. This is especially true during the opening or early middlegame phase, but does become more expensive in the late middlegame or endings with sliding pieces, especially queens.

On the Fly

Programmers like Joël Rivat [2], Robert Hyatt [3], Ed Schröder and Gerd Isenberg avoid or have abandoned incrementally updated attack tables and rely on the paradigm to process information if needed. A lot of nodes don't need the attack information at all, or only a small part of it. With all the hash tables, incremental update tends to do some unnecessary work, considering the update costs in "worst case" positions, f.i. queen endings, where one move change the attack information of many squares.

On the other hand, if attack tables are available, one should utilize the information as much as possible for a smarter search and evaluation to gain exponentially. Anyway, one has to be careful with too complicated data structures and update code.

Implementations

Classical Approach

The square centric classical approach with bitboards was used in Chess 4.5 and descibed by Larry Atkin and David Slate [4] . The incrementally updated attack tables, from which most move generation is done, are called *ATKFR* and *ATKTO*. *ATKFR* is a set of 64 bitboards which give, for each square, all the squares attacked by the piece, if any, that resides on the square. *ATKTO* (Square Attacked By) is the transpose of *ATKFR*, giving for each square, the locations of all pieces that attack that square. For instance the square E4 (T) is attacked by a black rook at E8, a black knight at F6, and defended by a white rook at E1 and a white pawn at D3 [5] :

```
attacks_to[E4]
 . . . . 1 . . .
 . . . . . . . .
 . . . . . 1 . .
 . . . . . . . .
 . . . . T . . .
 . . . 1 . . . .
```

```
. . . . . .
. . . . 1 . . .
```


Alternatives

There are several alternatives for keeping the square centric information what pieces attack each particular square.

Piece-Sets

A Square Attacked By bitboard aka *ATKFR* as possible union-set of multiple pawns and pieces of either side require intersections with piece bitboards, or bitscanned square lookups, to determine which pieces and how many attack or defend.

Based on a fixed piece-type and bit-position relation with usual material dispositions (for each side, no more than one queen, two rooks, one bishop per square color, two knights), 32-bit Piece-Sets already inherit the information which pieces (and how many of both sides) attack a particular square, one can even imagine a 16-bit lookup inside a 64KByte table to get an denser attack indicator/count byte for each color a lá Ed Schröder. MS-DOS IsiChess maintained an array of 64 32-bit piece-sets for every square, and an array of up to 32 attack-to bitboards for every piece. However working with piece-sets requires an additional indirection via a Piece-list to get the square of that piece.

Ed's lookup

As described by Ed Schröder in *Evaluation in REBEL* [6] , Rebel uses two board tables for both sides, one byte entry each, the three lower bits contain an attack counter, while the five upper bits indicate the presence of least one pawn or piece attacking/defending:

```
+------+------+------+------+------+------+------+------+
| BIT0 | BIT1 | BIT2 | BIT3 | BIT4 | BIT5 | BIT6 | BIT7 |
+------+------+------+------+------+------+------+------+
|      Number of     | PAWN |KNIGHT| ROOK | QUEEN| KING |
|      ATTACKERS     |      |BISHOP|      |      |      |
+------+------+------+------+------+------+------+------+
```

The information might be inaccurate in some cases since it loses some information if multiple pieces of one kind are involved. However, since SEE might be erroneous anyway due to pins, x-rays or overloaded pieces, Ed's scheme seems sufficient for practical purposes - and it is fast. Each byte (for both sides) can act as index inside pre-calculated three-dimensional table to perform an SEE by looking up a target piece or square, attack- and defend-byte:

```
char see_table [14][256][256];   // 14*64 K = 896 KByte

see = see_table[Piece][attackByte][defendByte];
```

While the counter might be updated incrementally, the piece indicators as possible union of multiple pieces (i.e. two knights and one bishop) is not that simple to update, thus Ed generates those tables in evaluation on the fly by scanning the pieces of the board.

Direction wise

An other alternative to incremental updated attack tables is motivated by direction wise fill algorithms like Kogge-Stone for sliding pieces, and that one may hide memory latencies from probes of the transposition table. Especially pawn attacks are cheap to determine on the fly, and likely reduce the set of capture targets of least valuable pieces defended by pawns, which are otherwise object of Quiescence Search or SEE.

See also

- Piece-Sets
- Bitboards
  Sliding Piece Attacks
  Square Attacked By
  Pieces versus Directions

Forums Posts

1995 ...

- Chess programming using bitboards by Joël Rivat, rgcc, August 18, 1995
- Attack Tables by Roberto Waldteufel, CCC, October 20, 1998

2000 ...

- Counting attacked squares: how? by Leen Ammeraal, CCC, January 24, 2002
- attacks_from[] and attacks_to[] info by Nagendra Singh Tomar, CCC, October 21, 2002
- Attack tables by Andreas Herrmann, CCC, November 20, 2002
- The Zappa Attack Table Code by Anthony Cozzie, CCC, May 05, 2004
- bitboards and incrementally updated attack tables by Eric Oldre, CCC, June 30, 2004
- Attack table by Anonymous, Winboard Forum, October 06, 2004

2005 ...

- Attack table musings by GeoffW, Winboard Forum, February 11, 2005
- incrementai updating for positionai evaiuation by Steven Edwards, CCC, March 27, 2008
- Piece attacks count by Marco Costalba, CCC, May 18, 2009 » Population Count

2010 ...

- Incrementally-updated attack map by Harm Geert Muller, April 21, 2014 » Incremental Updates

References

1. ^ The Game of War from Samuel Bak - represented by Pucker Gallery since 1969
2. ^ Chess programming using bitboards by Joël Rivat, rgcc, August 18, 1995
3. ^ Speed of Move Generator by Valavan Manohararajah, rgcc, August 15, 1995, post 5 by Robert Hyatt where he mentions on the fly generation with rotated bitboards
4. ^ David Slate and Larry Atkin (**1977**). *CHESS 4.5 - The Northwestern University Chess Program.* Chess Skill in Man and Machine (ed. Peter W. Frey), pp. 82-118. Springer-Verlag, New York, N.Y. 2nd ed. 1983. ISBN 0-387-90815-3. Reprinted (**1988**) in Computer Chess Compendium
5. ^ Rotated bitmaps, a new twist on an old idea by Robert Hyatt
6. ^ Evaluation in REBEL (hanging pieces) from How Rebel Plays Chess by Ed Schröder, also available as pdf

What links here?

Page	Date Edited
Andreas Herrmann	Nov 7, 2014
Arasan	Jan 6, 2018
Attack and Defend Maps	Nov 5, 2016
Bebe	Dec 23, 2017
Bitboards	Nov 14, 2017
BlackBishop	Nov 7, 2014
Board Representation	Dec 11, 2017
Bruja	Feb 3, 2015
CAPS	Dec 23, 2017
Check	Feb 1, 2018
Chess (Program)	Dec 22, 2017
Chess 0.5	Nov 20, 2016
Chest	Mar 3, 2015
Combinatorial Logic	Apr 6, 2017
Constellation	Oct 2, 2016
DanChess	Jun 17, 2012
Data	Nov 26, 2017
Deflection	May 21, 2011
Diablo	May 17, 2016
Dorpsgek	Jun 10, 2017
Double Attack	Oct 22, 2014
Encoding Moves	Mar 27, 2016
Eugen	Jan 7, 2016
Eugenio Castillo Jiménez	Jul 23, 2017
Eye Movements	Jul 22, 2015
Floyd	Sep 11, 2016
Golch	Dec 7, 2017
Gromit	Apr 5, 2017
Hagrid	Apr 12, 2016
Horizon	Mar 17, 2013
Incremental Updates	Sep 6, 2017
Interception	May 21, 2011
Interference	May 20, 2011
Joker NL	Sep 15, 2017
José Carlos Martínez Galán	Feb 19, 2017

Page	Date Edited
King Safety	Feb 14, 2018
Knight Cap	Nov 6, 2016
Legal Move	Feb 16, 2017
M-20	Oct 9, 2013
Mate at a Glance	Sep 24, 2014
Mediocre	Feb 27, 2015
MessChess	Mar 6, 2014
Mikhail Botvinnik	Jul 15, 2017
Mobility	Jan 17, 2018
Movei	Jan 7, 2016
MVV-LVA	Oct 26, 2017
Neural Networks	Mar 12, 2018
Node Types	Oct 22, 2017
Nullmover	Jul 21, 2013
Ostrich	Dec 28, 2017
Overloading	May 5, 2017
Pandix	Feb 4, 2017
Patzer	Apr 5, 2017
Perceiver	Nov 21, 2017
Peter Fendrich	May 19, 2017
Piece-Sets	Jun 9, 2017
Pioneer	Dec 23, 2017
Population Count	Sep 3, 2017
RDChess	May 3, 2013
Rookie	Jan 7, 2016
SEE - The Swap Algorithm	Jun 5, 2017
SnailChess	Nov 26, 2014
Snitch	Apr 7, 2016
SOMA	Mar 25, 2015
Spartacus	Oct 17, 2016
Spector	Nov 11, 2016
Square Attacked By	Jan 20, 2018
Square Control	Sep 15, 2016
Static Exchange Evaluation	Dec 14, 2017
Sunsetter	Jun 1, 2017
Tao	Jan 8, 2016
Vladimir Butenko	Aug 12, 2013
Woodpusher	Dec 12, 2016
Zach Wegner	Jan 7, 2016
Zappa	Oct 24, 2017

Up one Level