The Hoffman Reference Guide

Brent Baccala

February 24, 2009

Hoffman is a program to solve chess endgames using retrograde analysis, which is much different from conventional computer chess programs. Retrograde analysis is only useful in the endgame, runs very slowly, and produces enormous amounts of data. Its great advantage lies in its ability to completely solve the endgame. In a very real sense, a retrograde engine has no "move horizon" like a conventional chess engine. It sees everything. For those not up on Americana, the program is named after Trevor Hoffman, an All Star baseball pitcher who specializes in "closing" games. It was written specifically for The World vs. Arno Nickel game.

Hoffman uses XML extensively both for configuring its operation and for labeling the resulting tablebases. In fact, a completed Hoffman tablebase (typically in an `.htb` file) is just a `gzip`-ed file that contains an XML prefix followed by binary tablebase data (in a format specified by the XML). Basically, to operate Hoffman, you write an XML file that specifies the analysis you want done, then feed it to the program. As output, it produces a modified version of the XML input that includes binary tablebase data appended at the end.

1 Hoffman and Pawngen

A single Hoffman tablebase contains a fixed number and type of pieces, which can be placed onto different squares of the chessboard. For example, a "kqkp" tablebase typically contains all possible chess positions that can be formed using a white king and queen as well as a black king and pawn. It does *not* contain any positions that contain just a black king without a black pawn. Captures, promotions, and movements outside of a piece's allowed squares are handled by consulting other tablebases, called *futurebases*, which must already exist. It is also possible to handle such moves without futurebases by *pruning* them.

Creating and managing an interlinked set of tablebase control files can be quite daunting. Hoffman includes an auxiliary program called "pawngen" to aid in this process. Pawngen takes an Hoffman-format XML control file as input, considers all possible tablebases that can result, and outputs a directory full of correctly interlinked XML control files, ready for Hoffman, along with a UNIX standard makefile to construct them in the proper order. Pawngen currently accepts only a subset of Hoffman's XML syntax — no `location` attributes are allowed on any non-pawns, and pawns must have `location` attributes specifying a specific starting square followed by a plus sign. Pawngen does understand `futurebase` and `prune` elements, and will use them to truncate its analysis and thus limit the number of generated control files. Also, pawgen understands the `pawngen-condition` attribute on `prune` elements, which can be used to prune more selectively than otherwise possible.

Put simply, pawngen considers individual pawn formations and pawn moves (including captures and promotions), while allowing the other pieces to move freely around the board. Hoffman, on the other hand, can only consider one set of pieces at a time, requiring futurebases to handle captures and promotions. Furthermore, for Hoffman to compute the complicated changes in pawn formation that can result from various captures would require fairly unrestricted pawn locations that would produce prohibitively large tablebases.

Unlike Hoffman, which is written in C, pawngen is a Perl script, so you must have Perl installed (along with its `XML::LibXML` module) in order to use it. Calling Hoffman in the correct order on a directory containing hundreds of control files is a job for "make". Both Perl and make are fairly standard on Linux and other UNIX variants. On Microsoft systems, I've used the cygwin distribution (which includes both programs) successfully.

2 Parallel Processing with Hoffman

A Hoffman analysis can be quite compute-intensive. The program can be compiled to use POSIX threads (if available), with the number of threads specified at run-time using the `-p` option. The program is also designed to use multiple computers in parallel, all working simultaneously on an analysis. This is accomplished by breaking the analysis up into smaller pieces, each with its own XML configuration file. The primary support provided by the program is the ability to use URLs instead of filenames to reference tablebases, allowing tablebases to be stored on a server accessable over a network. A basic Perl CGI script (`hoffman.cgi`) is provided which, when installed on a web server and supplied with a directory full of Hoffman XML files, will hand them out in the proper order to Hoffman clients for processing.

3 Propagation tables

A Hoffman analysis can also be quite space-intensive. Since its memory utilization pattern is basically random, Hoffman will begin to swap dramatically and suffer a disastrous drop in performance once its working set size exceeds the machine's available memory. To alleviate this, the program can be operated in a mode where it fills a series of *propagation tables*, writing each one out to disk when full, then reads them back in sequentially during the next pass. Although less efficient than when the working set can be contained in memory, propagation tables allow the program to build tablebases of essentially unlimited size with no swapping and reasonable CPU utilization. This mode is activated at run-time by specifying the size of the propagation tables (in MB) with the `-P` switch.

Note: If you're using proptables, performance will be abysmal unless you specify a `modulus` attribute to the `index` element.

4 XML Syntax

The root XML element in a Hoffman tablebase is always `<tablebase>`. Its only attribute (`offset`) is added by the program, should not be supplied by the user, and indicates a hexadecimal byte-offset into the file where the binary tablebase data begins.

Within a `<tablebase>` the following elements may occur in the listed order (deprecated elements and attributes are not documented):

4.1 <prune-enable color="white|black" type="concede|discard"/>

Specifies which kinds of pruning elements will be allowed in this tablebase and its futurebases. Both attributes are required. `concede` means wins may be conceded to the named color; `discard` means moves by the named color may be discarded. At most one `prune-enable` can be specified for each color. No `prune-enable` element is required, however, no `prune` elements are allowed without one and no futurebases may possess additional `prune-enable` elements beyond those specified for the current tablebase.

4.2 <index type="naive|naive2|simple|compact|no-en-passant" symmetry="1|2|2-way|4|8|8-way" modulus="auto|*integer*"/>

The `<index>` element specifies the algorithm that will be used to compute the index numbers in the tablebase; i.e, the algorithm that will convert board positions into tablebase offsets and vice versa.

`naive` uses 2^{6n+1} indices to store positions for n pieces. It assigns a single bit for the side-to-move flag, then assigns 6 bits to each piece, which is used to encode a number from 0 to 63, indicating the piece's position on the board.

`naive2` Differs from `naive` in its handling of multiple identical pieces, which it stores as a base and an offset, thus saving a single bit. Currently, only pairs of identical pieces are handled; a fatal error will result if there are more than two identical pieces.

`simple` Like `naive`, but only assigns numbers to squares that are legal for a particular piece. Slower to compute than `naive`, but more compact for tablebases with lots of movement restrictions on the pieces.

`compact` A combination of the delta encoding used for identical pieces in `naive2`, the encoding of restricted pieces used in `simple`, plus a paired encoding of the kings so they can never be adjacent.

`no-en-passant` An enhancement of `compact` that uses the paired encoding scheme for pawns restricted to the same file. Since they can never pass each other, we can encode them as if they were an identical pair, then assign their colors in the same order they were originally specified. En passant significantly complicates this and can not be handled with this scheme.

The optional `symmetry` attribute can be used to encode multiple positions using a single entry, but its utility depends upon the exact analysis being done. A tablebase with no pawns and no movement restrictions can be encoded with `8-way` (alias `8`) symmetry, since the board can be rotated about a horizontal, vertical, or diagonal axis without affecting the behavior of the pieces. A tablebase with pawns can utilize at most `2-way` (alias `2`) symmetry, since only a reflection about a vertical axis preserves piece behavior. A tablebase with restrictions on the positions of the pieces (say, frozen pawns) can not use any symmetry at all (`1`). Not all symmetries are compatible will all index types; for example, 8-way symmetry can not be used with `naive` or `naive2` index types. **Default:** *no symmetry*

The optional `modulus` attribute, which if specified should be either `auto` or a prime number (the program complains if it's composite), indicates that the computed index should be inverted in a finite field (modulo the specified number) to obtain the actual index. While time consuming, this step has the effect of shuffling the indices in a pseudo-random fashion, and should be used if proptables are in use in order to optimize the operation of the library sort. This also produces larger tablebases, since a pseudo-random distribution of mating positions impedes the operation of the `gzip` compression algorithm. `auto` simply rounds the highest index up to the next prime number; there really is no reason anymore to specify a specific prime. **Default:** *no inversion*

4.3 `<format> ... </format>`

This optional element specifies the format of the tablebase entries. It has no attributes, and must contain exactly one of the following elements:

`<dtm bits="8|16"/>` specifies a *distance to mate* metric occupying either one or two bytes. Zero is used for draws, -1 is used for positions where the moving side is checkmated, and 1 is used for positions where the moving side can capture the opposing king, so a one byte dtm can record mate-in distances up to 126. A two byte dtm has no such (practical) limitation.

`<basic/>` specifies a *bitbase* where two bits are used for each position, and no distance information is stored — only an indication of the ultimate outcome (win, lose, or draw). Such a format is more compact and requires less time to generate, but requires more effort to use, since care must be taken to avoid loops when following winning lines.

`<flag type="white-wins|white-draws"/>` specifies a *bitbase* where only a single bit is used for each position.

Default: 8-bit DTM.

4.4 `<piece color="white|black" type="king|queen|rook|bishop|knight|pawn" location="`*string*`"/>`

Multiple `piece` elements are used to specify the chess pieces present in the tablebase. `color` and `type` are required and should be obvious. The ordering of `piece` elements is significant in that it directly affects the index algorithm, but there is no user-visible effect of the ordering.

The optional `location` attribute restricts the board positions available to this piece. It should be a list of squares, in algebraic notation, on which the piece is to be allowed. A single square results in a completely frozen piece. In addition, pawns may use an additional syntax consisting of a single starting square followed by a plus sign, indicating that the pawn may move forward as far as possible. This can be used, for example, to locate a black pawn on `"a7+` and a white pawn on `"a2+"`, indicating that both can move forward, but they can not "pass" each other.

4.5 `<futurebase filename="`*string*`" url="`*string*`" colors="invert"/>`

One or more futurebases may be specified with this element. Either a `filename` or a `url` may be specified (not both) to locate a futurebase, which must be another Hoffman tablebase. It must be related to the current tablebase in one of the following ways:

It has exactly the same piece configuration as the current tablebase, and corresponds to movement by one of the restricted pieces, i.e, the current tablebase has a white pawn frozen on `e4` and the futurebase has a white pawn frozen on `e5`.

It has exactly the same piece configuration as the current tablebase except that a single piece is missing, i.e, a capture occurred.

It has exactly the same piece configuration as the current tablebase except that a single pawn has been replaced with a knight, bishop, rook or queen, i.e, a pawn promoted.

It has exactly the same piece configuration as the current tablebase except that a single pawn has been replaced with a knight, bishop, rook or queen, and a single non-pawn of the opposite color has been removed, i.e, a pawn captured and promoted in the same move.

The option `colors="invert"` attribute may be specified to indicate that the piece colors of the futurebase should be inverted as it is processed. This precludes the need to calculate, say, a tablebase with a white queen and a black rook as well as a tablebase with a black queen and a white rook. The first may be used (with this option) as a futurebase to calculate a tablebase with two white rooks and a black queen.

Note: Any futurebase `prune-enable` elements must be a subset of the current tablebase's `prune-enable` elements.

Note: The only `url` scheme currently supported for this element is `ftp`.

4.6 <prune color="white|black" move="*string*" type="concede|discard" pawngen-condition="*perl-expression*"/>

Futuremoves not handled by specifying futurebases must be pruned using one or more of these elements, or an error will result. All three attributes are required. The `move` is specified using regular expression syntax to match a move in a subset of standard algebraic notation. All of the following strings are examples of legal `move` strings in a `prune` element: `Pe5`, `P=Q`, `RxQ`, `PxR=Q`. The following regular expressions would all match `Kd4`: `Kd?`, `K?4`, `K[a-d]4`, `K*`. The `type` attribute specifies what should be done with matching moves: treated as wins for the moving side (`concede`), or completely ignored (`discard`). If multiple `prune` elements match a particular move, it is a warning if they have the same `type`, a fatal error if their `type`s differ.

A single `prune` element may be specified with `move="stalemate"` and `type="concede"`. In this case, the `color` attribute indicates to which side stalemates should be conceded as wins.

The optional `pawngen-condition` attribute is not allowed directly by Hoffman, but can be present on control files input to pawngen. It specifies a Perl expression to be tested on each position considered by pawngen — i.e, every distinct pawn formation with an associated set of non-pawn pieces. The prune statement will be included in the resulting Hoffman control file only if the Perl expression evaluates true. The Perl expression is evaluated in the following context: `@white_pieces` is an array of one-character strings listing the non-pawn white pieces, i.e, `("K", "Q")`; `@white_pawns` is an array of two-character strings listing the board squares of the white pawns, i.e, `("a4", "c6")`; and likewise for `@black_pieces` and `@black_pawns`. This context may change slightly in future releases.

Note: If a `prune` element is specified for a futuremove handled by a futurebase, then the futurebase takes precedence. However, this case is handled by tracking every futuremove in every position, so it is possible to specify futurebases that handle a subset of the possible futuremoves, then use `prune` elements to handle the rest by default.

Note: `prune` elements are only allowed if they match a `prune-enable` element. If no `prune-enable` elements were specified, then no `prune` elements will be permitted.

4.7 <generation-controls> ... </generation-controls>

This optional element has no attributes and contains one or more of the following sub-elements, in no particular order:

4.7.1 <output filename="*string*" url="*string*"/>

At most a single `output` element should be used, with either a `filename` or a `url` (but not both), to specify where the finished tablebase should be written.

Note: The only `url` scheme currently supported for this element is `ftp`.

4.7.2 <completion-report url="*string*"/>
<error-report url="*string*"/>

Optionally, the tablebase's XML prefix (without the tablebase data) can be written to a URL upon either a successful or error termination of the program. This capability (along with the ability to be read and write tablebase URLs) is intended to aid the construction of Hoffman tablebases using a distributed cluster.

Note: In the event of an error termination, every attempt will be made to add `<error>` elements to the XML indicating the cause of the problem.

Note: The only `url` scheme currently supported for these elements is `http`.

4.7.3 <entries-format> ... </entries-format>

This optional element controls the internal format used to store a tablebase during its construction. It contains a number of entities, each corresponding to a structure field, all of which admit at least the attributes `bits` and `offset`, which specify, respectively, the number of bits occupied by the field and the field's offset relative to the beginning of a tablebase entry. `offset` is optional and, if not specified, will be computed using an algorithm to assign fields to empty slots, though if `offset` is specified in one element, it must be specified in all of them. `bits` is usually required, except for single-bit-only fields. The total number of bits required must be a power-of-two byte boundary, i.e, 8, 16, 32, 64. The entities are:

`dtm` Distance-to-mate. Required to generate a tablebase with a `dtm` `format`; otherwise useless and optional. Can be used with any number of `bits` (unlike the `dtm` element in the tablebase `format`, which can only have 8 or 16 `bits`), but a fatal error will result during tablebase propagation if this field is too small.

`movecnt` Required. Used to count the number of possible moves from a given position. Has four reserved values, and must be able to count down to zero, so an n-bit `movecnt` allows positions with up to $2^n - 5$ moves. If 8-way symmetry is in use, then many positions require their moves to be counted twice, effectively halving that number. If positions exist with too many possible moves to fit into `movecnt`, a fatal error will result during tablebase initialization.

`locking-bit` Single-bit-only field. Optional, but recommended if running multi-threaded. Allows individual tablebase entries to be locked. Without it, a global lock must be used to regulate access to the entries table, which can adversely affect performance, but probably not as much as doubling the size of the entries table if that is the only way to create a free bit.

Default:
```
<entries-format>
   <dtm bits="8" offset="0"/>
   <movecnt bits="7" offset="9"/>
   <locking-bit offset="8"/>
</entries-format>
```

Note: Changing `entries-format` requires the program to be recompiled with a different `formats.h` file, which the program will print as it terminates with a fatal error.

Example: A good example of how to use this element is found in the standard `kppkp` tablebase, which has a maximum distance to mate of 128, and therefore requires a 9 bit `dtm` field (unfortunately, there's no way to know this in advance). This can be achieved within a 2-byte `entries-format` by stealing a bit from the `movecnt` field, leaving it with 6 bits for a possible $2^6 - 5 = 59$ movements. Since a king can have no more than 8 movements, and a pawn can have no more than 12 (on the seventh rank it has two possible captures and a forward movement, times four possible pieces it can promote into), a king and two pawns can have no more than 32 movements (less, actually, since no capture-promotions are possible with only a pawn and a king on the opposing side), so a 6-bit `movecnt` works fine. On the other hand, this alternate format would not work on the `kqqkq` tablebase, since a queen can have up to 28 movements, so a king and two queens can have around 74 movements, which gets doubled to 148 since we use 8-way symmetry.

4.7.4 `<proptable-format> ... </proptable-format>`

This optional element controls the internal format used to store proptables, and is structured like `entries-format`. The possible entries here are:

`dtm` Exactly as specified in the `format`.

`PTM-wins-flag` Single-bit-only field. Similar to the `flag` field in `entries-format`, but has a slightly different interpretation (indicates if *player to move* wins, not white). No `type` attribute. Should be specified instead of a `dtm` field if a bitbase is being constructed.

`movecnt` Similar to the same field in `entries-format`, but has no reserved values and doesn't have to hold a complete move count (only the number of moves being propagated). Currently has **no check** for overflow!!

`index-field` Holds the index number being propagated. See the `index` section for more information about how index numbers are computed. Currently has **no check** for overflow!!

`futurevector` Only used during the first back-propagation pass, to track which futuremoves have been handled as futurebases are back-propagated. Must be large enough to hold a single bit for each possible futuremove from a position. Hoffman will die (early) with a fatal error if this field is not large enough.

Default:
```
<proptable-format>
   <index bits="32" offset="0"/>
   <dtm bits="16" offset="32"/>
   <movecnt bits="8" offset="56"/>
   <futurevector bits="64" offset="64"/>
</proptable-format>
```

Note: Changing `proptable-format` requires the program to be recompiled with a different `formats.h` file, which the program will print as it terminates with a fatal error.

4.8 <tablebase-statistics> ... </tablebase-statistics>

This element is added by the program and should not be specified in the input. It contains statistics relating to the finished tablebase.

Element	Interpretation
indices	Total number of entries in the uncompressed tablebase
PNTM-mated-positions	Total number of positions in which *player not-to-move* is mated; i.e, illegal positions in which a kind can be immediately captured
legal-positions	Total number of legal positions; i.e, total number of entries, minus illegal entries where two pieces occupy the same space, minus PNTM-mated positions
stalemate-positions	Stalemate (not draw by repetition) positions
white-wins-positions	Positions from which White can force a win
black-wins-positions	Positions from which Black can force a win
forward-moves	Total number of forward moves from positions in this tablebase (including futuremoves)
futuremoves	Total number of forward moves from positions in this tablebase into futurebases or pruned
max-dtm	Largest *distance to mate* of all positions in this tablebase
min-dtm	Smallest *distance to mate* of all positions in this tablebase, i.e, a negative number indicating the longest forced loss

4.9 <generation-statistics> ... </generation-statistics>

This element is added by the program and should not be specified in the input. It contains statistics relating to the program run that generated the tablebase.

Element	Interpretation
host	Hostname of system that generated the tablebase
program	Name and version of the program that generated the tablebase
args	Command line used for the generation run
start-time	Time the program run initially started
completion-time	Time the program run finally ended
user-time	CPU time used by the run in user space
system-time	CPU time used by the run in system calls
real-time	Wall clock time used by the run
page-faults	Number of times the program had to wait for a memory page to be swapped in from disk
page-reclaims	Number of times the program reclaimed a page from the free list; this will typically be program instruction pages
proptable-writes	If proptables are in use, the number of proptables written to disk
proptable-write-time	If proptables are is use, the total real time required for all proptable writes
pass	Per-pass statistics, including real-time and user-time

5 Some Confusing Error Messages

5.1 Doubled pawns must (currently) appear in board order in piece list

Currently, doubled pawns using "plus" locations (ex: `location="a2+"`) on the same file must have their `piece` elements listed in the XML in the order that the pawns appear on the board, counting in algebraic notation from row 1 to row 8. I mean, row 2 to row 7.

5.2 Piece restrictions not allowed with symmetric indices (yet)

You can't specify an `index symmetry` attribute and also specify `piece location` attributes, even if the restrictions on the piece locations might be compatible with the requested symmetry.

5.3 Non-identical overlapping piece restrictions not allowed with this index type

For the `naive`, `naive2`, and `simple` index types, you can't specify two identical pieces with different `location` restrictions unless those restrictions are completely distinct. For example, you can't have a free white rook and another white rook restricted to the a-file. If you think about it, this situation would allow the rooks to "trade places" — both could move to the a-file and then either one could move off. The simpler index types can't handle this situation. You could, however, have a white rook restricted to the a-file and another restricted to the d-file (or use a more sophisticated index type, like `compact`).

5.4 More than two identical pieces with overlapping move restrictions

Identical pieces (say, two white rooks) can be swapped without changing the position, and the program has to take this into account. If there were, say, three identical pieces, we'd have to consider a more complex set of permutations, and the program currently doesn't do this. So, unless the pieces have non-overlapping move restrictions (i.e, they can't be swapped), we're currently limited to two identical pieces, and thus can't handle something like `kpppkr`.

5.5 Attempting to initialize position with a movecnt that won't fit in field!

The `movecnt` field specified in `generation-controls` wasn't big enough, and note that the default value might not be big enough!

5.6 Default/specified proptable/entries format incompatible with compiled-in format

Changing the formats in `generation-controls` requires recompiling the program with a different `"formats.h"` file.

5.7 Futurebase doesn't match prune-enables!

Remember that futurebase `prune-enable` elements must be a subset of the current tablebase's `prune-enable`s.