# 1 A comparison of Rybka and IPPOLIT

The purpose of this document is to list some similarities and differences between Rybka and IPPOLIT. The information regarding the former was obtained via a painstaking analysis via debugging tools. I do not claim to understand every facet of Rybka, but do consider myself sufficiently expert to produce this document.

## 1.1 Timeliness of this document

Although the early history of IPPOLIT is unclear, it seems that May 2009 is the first time that a direct claim regarding its parentage was made by Vasik Rajlich. As the classical length of insordescence is one year, I find no reason not to discuss his claims freely.

## 1.2 Scope of this document

I will attempt to stick with verifiable evidence from the relevant programmes, and demur any discussions of the multitude of sideshows that have arisen. I have tried to emphasize *code* similarities rather than *idea* similarities, as the latter are harder to pinpoint.

## 1.3 Versions

Rybka means Rybka 3 Chess, and IPPOLIT means code akin to the original `IPPOLIT.c` and/or `IPP_ENG.c` (though I didn't try to understand the Russian). My understanding is that I am using the "contempt-free" values from Rybka. I have not looked at the "Human" and "Dynamic" versions of Rybka 3 in depth, but have done a few experiments that indicate that their discussion is not overly relevant here.

# 2 Overview

The programmes will be compared and constrasted on the major parts that go into a chess engine. The relevant major headings include Board Representation, Search, and Evaluation, each of which has various subcomponents. I consider that it should be up to the reader to determine the relative significance. Two appendices are included to aid in comparing the particularities of evaluation and search.

# 3 Board representation and internal structures

Rybka and IPPOLIT both utilize Hyatt's rotated bitboards. I see no evidence (such as ordering of variables) that IPPOLIT has used decompiled code of Rybka to achieve this.

### 3.1 Bitboard arrays

Both have a large number of bitboard arrays that are used for convenience. For example: squares which a bishop could attack from a square; squares that should be free from pawns for an open file; masks for isolated files, etc. The overlap is large, but I don't know whether it is notable. I do not see any "suspicious" array that Rybka contains which then appears in IPPOLIT. In fact, the more curious arrays of Rybka do not recur in IPPOLIT, even in surrogate form. IPPOLIT largely generates in its own arrays on startup, but this is probably unimportant.

### 3.2 Move encoding

Both Rybka and IPPOLIT use 32 bits integers for move encoding. The lowest 12 bits in both are `from`|`to`. The 12th and 13th bit in Rybka are for promotion, and the 14th for en passant, with the 15th being for if a quiet move is check. There is no encoding for castling, as that is detected with (say) `e1g1` by noting that a king moved 2 squares. IPPOLIT uses bits 12-14 for promotions (4-7), *en passant* (3), and castling (1). The 15th bit is used (following Rybka) for whether a quiet move is check, but also to help with history values when already in check.

The upper 16 bits have a capture score (MVV/LVA essentially) or a history score, or a "positional gain" score. Rybka uses a linear fit for capture scores, while IPPOLIT has a handmade table and considers promotions (both `Q` and `N`) in a different manner. These capture scores only take up part of the 16 bits, and the other bits are marked as to whether a capture is always SEE. IPPOLIT uses bit 21 (via `2<<20`) when a capture is good (like `PxN`), and bit 20 when a capture is equal (like `PxP`). With Rybka I think it is bits 16-19 that are used.

The history score for non-capture moves is `piece`|`to` for both. Rybka uses two arrays, depending on whether the scout value is close to the evaluation, while IPPOLIT only has one, and it simply ignores a fail-low history subtraction if the scout and evaluation differ by too much.

### 3.3 Piece encoding

Rybka has `OoPpNnBbRrQqKk` as its piece order, where `Oo` is occupied squares, and capital letters are White. IPPOLIT has `OPNKBBRQopnkbbrq`, where the first `B` is the light-squared bishop, and the second is the dark-squared. This difference in encoding must be kept in mind at all times during a comparison, as it forces many things to be different by necessity. So to make a proper comparison, some "translation" from one piece-set to the other must be made.

### 3.4 Material tokens

Both Rybka and IPPOLIT dynamically update material indexes during the `make_move` routine. These are then used in evaluation to get the material value (though IPPOLIT has the actual value, while Rybka scores the adjustment from the 1-3-5-10 standard). Rybka does split the bishops in this indexing, but not

at the bitboard level as with IPPOLIT. Rybka has 64 bits for this token, while IPPOLIT has 32 bits, though I'm not sure how often Rybka uses some of the fields. IPPOLIT reserves the highest bit to handle promotions (such as multiple queens), but Rybka does not seem to have any method to handle overflow.

### 3.5 Material imbalances

Rybka generates a table of material imbalances using a three-stage interpolation. IPPOLIT uses a 4-stage interpolation, and the 1-3-6 weights (for minor-rook-queen) with IPPOLIT are not the same as in Rybka. Rybka also has some extra weightings that IPPOLIT does not, regarding mixed minors. Both have some affinities for knights and rooks corresponding to the count of pawns.

Both have a token for "winning chances" that (for instance) is reduced in an endgame of opposite coloured bishops, etc. The values for IPPOLIT are a bit different, as (for instance) they vary with how many pawns are left, unlike in Rybka. IPPOLIT also embeds into this token whether a nullmove should be tried. Finally, both have a "weight" for the endgames with zero or one pawns (like `RB` vs `R`), and here Rybka's code/values seem more structured than IPPOLIT's, as the latter is more akin to a bunch of special cases. Rybka seems to have these weights on a scale of 0-16, while it is 0-10 with IPPOLIT.

### 3.6 Static Values (PST)

IPPOLIT generates these by a formula on startup. With aid from Larry Kaufman, it seems that Rybka's originally came from similar formulæ (and indeed Fruit did something similar already). The specific values are somewhat similar, though one needs to consider the material weighting for a complete comparison. More about the exact static values appears in an appendix,

Both Rybka and IPPOLIT use one 32-bit integer to keep opening/endgame values, but Rybka has an additional `0x80008000` in it presumably to aid with over/underflow at the 16-bit boundary. I will say more about this in Evaluation. Also, Rybka uses millipawns, while IPPOLIT has centipawns. Again this needs to be kept in mind when trying to make a comparison.

## 4 Board updating and scanning functions

Both Rybka and IPPOLIT have a variety in this genre, such as move generation functions, make/unmake, and SEE. The most notable similarity is that both split every such function into two distinct black/white function pairs. Rybka has a few more move generators, and an extra SEE.

### 4.1 Make and Undo

It's hard to know what would be considered "similar" here. Due to the various differences in board representation, there are various differences in the code.

I'm not sure what to say beyond that, as the make/undo functions are fairly formulaic in any event. I'll expound, and illustrate many of the difficulties in comparsion, by listing the `make_white_castle` code (the rook part) for Rybka:

```
0x482a00:  xor    %eax,%eax
0x482a02:  cmp    $0x6,%ecx                  # if to is G1
0x482a05:  mov    %eax,2392757(%rip)         # 50move = 0
0x482a0b:  jne    0x482a84                ###### then do Q-side ooo
0x482a0d:  xorq   $0xa0,2392584(%rip)        # WhOcc ^= F1 | H1
0x482a18:  xorq   $0xa0,2392637(%rip)        # WhRook ^= F1 | H1
0x482a23:  xorq   $0xa0,2392698(%rip)        # Occupied ^= F1 | H1
0x482a2e:  xorq   $0x9,2392682(%rip)         # Right45 ^= F1 | H1
0x482a36:  addl   $0x5a0000,2392712(%rip)    # Static += 90op+0eg
0x482a40:  mov    %eax,2392310(%rip)         # Board[H1] = Empty
0x482a46:  mov    $0x80008000,%rax
0x482a50:  movl   $0x8,2392282(%rip)         # Board[F1] = WhRook
0x482a5a:  xor    %rax,2392655(%rip)         # Left90 ^= F1 | H1
0x482a61:  mov    $0x800100000,%rax
0x482a6b:  xor    %rax,2392614(%rip)         # Left45 ^= F1 | H1
0x482a72:  mov    $0xd8b3287ea544969,%rax
0x482a7c:  xor    %rax,2392653(%rip)         # Hash ^=  WRf1 | WRh1
0x482a83:  retq                           ###### below is Q-side
0x482a84:  xorq   $0x9,2392468(%rip)         # WhOcc ^= A1 | D1
0x482a8c:  xorq   $0x9,2392524(%rip)         # WhRook ^ = A1 | D1
0x482a94:  xorq   $0x9,2392588(%rip)         # Occupied ^ = A1 | D1
0x482a9c:  xorq   $0x201,2392561(%rip)       # Left45 ^ = A1 | D1
0x482aa7:  xorq   $0x10000400,2392558(%rip) # Right45 ^= A1 | D1
0x482ab2:  addl   $0x820000,2392588(%rip)    # Static += 130op+0eg
0x482abc:  mov    %eax,2392158(%rip)         # Board[A1] = Empty
0x482ac2:  mov    $0x80000080,%eax
0x482ac7:  movl   $0x8,2392155(%rip)         # Board[D1] = WhRook
0x482ad1:  xor    %rax,2392536(%rip)         # Left90 ^= A1 | D1
0x482ad8:  mov    $0xaaaff37267ceded3,%rax
0x482ae2:  xor    %rax,2392551(%rip)         # Hash ^= WRa1 | WRd1
0x482ae9:  retq
```

One can compare the IPPOLIT code, and there are similarities in ordering (`WhOcc`, `WhRook`, `Occupied`, `Left90`,`Left45`, `Right45`, `Static`, `Hash`, `Board[H1]`, `Board[F1]`), but not a complete copy; also, the C code appeared in `move_do_castle` of `utils.c` in Strelka in any case. IPPOLIT doesn't set the 50-move count to zero when castling, while Rybka does.

Both Rybka and IPPOLIT only check for stopping/input in `make_white_move`, and (I think) they both do so every `0x1000` (or 4096) moves, though the checking mechanism is different. Rybka is actually counting the number of times white `make_move` is called via subtracting down from `0x1000` and then incrementing a second count when it hits zero, while IPPOLIT is just counting

`white_make_moves` as a 64-bit integer, and checking whether its low 12 bits are zero (so that 4096 divides it). Rybka divides this count of white make-moves by 7 to get a "node" count, while IPPOLIT additionally counts black make-moves and null make-moves, and returns the total as the node count.

## 4.2 Move generation

Both have the standard generators for captures (and promotions), quiet moves, quiet checks, and check evasions. Rybka sees to have one than one generator for some of these. Both also have a move generator for moves that might be expected to gain a lot positionally. Rybka has a couple of these (IPPOLIT has one), and uses various masks for exclusion in them, while IPPOLIT does not.

### 4.2.1 Captures/Promotions

For captures/promotions, Rybka also computes pins/xrays (IPPOLIT does this in evaluation), and Rybka includes underpromotions (at least if a flag passed as a function argument is set), while IPPOLIT only includes knight underpromotions that give check (Rybka does also, even when the flag is not set). IPPOLIT does *en passant* first, while Rybka does it after other pawn captures. Both use the order: left pawn captures, right pawn captures, knights, bishops, rooks, queens, kings, and pawn promotions (including captures).

### 4.2.2 Noncapture moves

For noncapture moves, both keep track of whether the move gives check directly (xrays are handled differently), but IPPOLIT only precomputes the orthogonal/diagonal arrays (from the king's square) if a piece of the same sort is available. Both generate castling (kingside then queenside) first, but then Rybka follows the order `PNQBRK`, while IPPOLIT has `QRBKNP`. Of necessity, IPPOLIT has underpromotions (including captures) here. IPPOLIT has to do some chasing with its light/dark bishops to get the correct history values. To state the obvious, IPPOLIT considers bishop underpromotions, while Rybka does not.

### 4.2.3 Check evasions

For check evasions (at least the "main" one for Rybka), the structures are again similar but different. IPPOLIT has a "pieces giving check" array from evaluation, while Rybka has to compute this. IPPOLIT uses the 15th bit of the move to indicate that a checking move is not a capture, and in that case, a history value is appended to it later. I got confused as to what Rybka is doing, but the method seems to be to just set history values, and then later make a change for captures (and hash/killer moves; IPPOLIT also recognises these in the search). The method of generation seems to differ somewhat here, as IPPOLIT would catch "pawn captures that interpose (but not capturing the checking piece)", if such things existed. Other than this difference in handling pawns, both first do king moves (in conjunction with double check), then pawns, then knights, but

then IPPOLIT does `BRQ`, while Rybka does diagonal moves and then orthogonal moves.

#### 4.2.4 Quiet checks

For quiet checks (I'm not sure which is the "main" one for Rybka), IPPOLIT first uses its list of xray pieces (from eval), to generate discovered checks (`PNBRK`), then does (non-promoting) pawn captures (if the captured piece was not in the target of the capture generation), then `QRBN` moves with SEE checked, and finally some pawn pushes (more below).

For Rybka, method one uses a list of pieces that can check (computed in move generation). However, first knight checks are done, but sometimes only those that fork something, or the opposing king has no flee square, and/or a few other conditions with SEE. Then the possible-checkers are considered (`BSF` order), and the list of xrays computed at the same time. These are then looped over in `BSF` order (internally the code splits as `PBNRK`). Finally some pawn pushes are done. I don't see where pawn captures are done here, but maybe I missed it. The second method for Rybka does knight checks with simpler conditions than before, then does diagonal pieces, and then orthogonal ones. With the latter two, xrays are generated at the same time. All pawn pushes (and previously omitted captures) that give check are then generated, though a pawn-push needs to pass SEE.

Coming back to the pawn pushes and captures in method one, Rybka has a nontrivial condition involving whether the king can flee, or whether the pawn push is a fork, etc. IPPOLIT doesn't do this, and has a "pawn is guarded" condition (to avoid SEE if possible) that I don't see in Rybka. The only "curious" thing is that both Rybka and IPPOLIT check if the opposing king is on its 4th-8th rank before looking at pawn pushes, though I guess this is an "obvious" condition when you think about it (and the mask is combined with either `b-h` or `a-g` files, depending on whether left/right checks are being considered).

#### 4.2.5 Positional gain generators

Finally we get to the generators that only list moves with a sufficiently high "positional gain" either possible and/or likely – these generators are used slightly differently in search by Rybka and IPPOLIT in any case, but I will compare them here. Rybka has two of these, one of which only considers knight then pawn then king moves, all subject to masks (which can vary by game phase) on the to/from squares. The other Rybka function has a mask only for the knights, and considers `NPKBQR` (in order). For IPPOLIT, the order is `PNBBRQK`. The idea of these functions is to generate quiet moves, but only record those whose "positional gain" is large. Both place this "positional gain" in the upper 16 bits of a 32-bit move, though Rybka does this in general with quiet moves (I think), and IPPOLIT only in this special search (which leads to IPPOLIT accessing the same information later, directly from the array).

### 4.3 SEE

Rybka seems to have two SEE functions, but one of them is quite close to the normal version. I will remain silent about the other. It is hard to say what I should expect to be different between Rybka and IPPOLIT here, as the method is rather straightforward (and appears in Strelka). IPPOLIT has values like 6174, 12345, 23456, 12345678 for its "large" integers, and Rybka does not (it has 2001 and 2999 as with Strelka). IPPOLIT has a margin of $-60$ appearing in SEE, whil this is just 0 in the main Rybka SEE function. IPPOLIT uses 100,325,500,975, while Rybka uses 1,3,5,10.

Both Rybka and IPPOLIT use information about pins in SEE, but it comes from different sources. Rybka gets it from capture generation (and passes it to SEE as an argument), while IPPOLIT gets it from evaluation/mobility directly from an evaluation structure, and also tries to determine if the move passed to SEE undoes the pin.

## 5 Tables

### 5.1 Pre-evaluation

Rybka has a large amount of pre-evaluation, depending on the game phase (like Fruit, with 1+2+4 weighting for minor+rook+queen). About 20-25 parameters are controlled, including pruning margins, lazy margins, etc. IPPOLIT has no pre-evaluation that I can see.

### 5.2 History

Both Rybka and IPPOLIT have 16-bit history values based on piece-to, though Rybka only uses 15 bits. The update mechanism is similar, in that it uses a "sigmoidal" function to preclude overflow. IPPOLIT updates `H` to `H-(H*depth)/256` for bad moves, and `H` to `H-(0xff00-H*depth)/256` for good moves. Rybka has different values for `0xff00` and 256 here. Rybka also keeps a list of 4 killer moves (2 with IPPOLIT), though I'm not sure the latter two are ever used.

Rybka actually has two history tables, the second of which is used in certain circumstances when the scout value and the evaluation differ by a lot. IPPOLIT doesn't update history for "bad" moves when these differ by too much. The starting point for history scores in IPPOLIT is `0x800`, which is the not the same as Rybka (nor is it merely twice as much).

### 5.3 Positional Gain

Both Rybka and IPPOLIT have a "positional gain" table indexed by piece and to-from that tries to guess what the positional gain (or the maximal positional gain) from a move should be. This is updated after evaluation. Upon computing the position evaluation (and the move is not a capture), there are two possibilties: if the positional part of the evaluation is larger than the stored value,

the stored value is set to the positional part of the evaluation; otherwise the stored value is decremented by 1. Rybka's method is a bit different (different cutoffs, and scaling), though in the same vein. Rybka also only has an array of 7x64x64 entries here, re-using the same array for multiple pieces (for instance, the to/from for knights and rooks already distinguishes them), while IPPOLIT just has the 16x64x64 you might expect.

Rybka actually has a number of these positional gain tables, updated in various ways, though the other only appear to have some (minor) application in lazy conditions, if they aren't just totally ignored. IPPOLIT has only one positional gain table, and it does not appear in the lazy conditions. Rybka's stored values for positional gains are only 8 bits wide, while IPPOLIT's are 16. Rybka starts the positional gains at some values from a huge table (I have no idea from where it comes), while IPPOLIT has them all be 0 to begin.

### 5.3.1 A Smoking Gun?

IPPOLIT peculiarly has a leftover condition in the updating of positional gain, in that it first checks whether a move is a capture, and if not, it then checks whether it is a capture (again!) and the captured piece is a pawn; the latter is exactly a condition (among many) in Rybka's code, though it does appear in the same place (at the start). Also, Rybka does (many) other things here when a move is a capture, whereas IPPOLIT does nothing in this case.

This "could" be harmless: for instance, the IPPOLIT testers might have been just trying many different combos here, and didn't make the last deletion of redundant code. Perhaps humourously, this "copied" code (if it is that) never actually does anything in the end...

## 5.4 Hash

Rybka's main hash entries are 64 bits. IPPOLIT's are 128. Rybka uses 7 bits for depth, 3 bits for age, 12 bits for a move, 10 bits for a score, 4 bits for flags, and 28 bits of Zobrist value. IPPOLIT follows something more akin to Fruit, with 8-bit boundaries. The eviction criterion with ageing is different: Rybka counts age to 8 (by multiples of 128 I think), while IPPOLIT counts to 256 (by ones). The formula for combining depth with age is also not the same. IPPOLIT has the pedantic idea of changing the age of a zeroed entry (when an exact entry displaces bounds). The specific Zobrist values for piece-square combos are generated by IPPOLIT on start-up (using some randomised function), while they are simply read from a table in Rybka.

### 5.4.1 Main-hash functions

Rybka seems to have a couple more hashing functions, though they all serve the same purpose in the end. IPPOLIT splits the hash-writing functions by lower, upper, lower-ALL, upper-CUT, exact. Rybka on the other hand does not split CUT/ALL into separate functions, and so passes an argument to the

lower/upper hashing function to tell it whether/if the node is CUT/ALL. Both keep 4 entries in each compartment, with similar overwrite rules.

#### 5.4.2 Pawn hash

The pawn-hash entries in Rybka are 136 bytes, while IPPOLIT uses 64. I'm not sure Rybka uses all 136 bytes, but the saved information differs in any event (Rybka keeps rotated bitboards of pawns, for instance). Both have an evaluation hash, which has 32-bit entries in Rybka, and 64-bit entries in IPPOLIT. (In passing, I can note Rybka seems to contravene the UCI philosophy by failing to reset this evaluation hash when `ucinewgame` is invoked). There is a "pv hash" for IPPOLIT (to try to avoid moves being overwritten when the PV is reconstructed), which does not appear in Rybka.

Both use pawn-hash in conjunction with king position, so that it is really a king+pawn hash. However IPPOLIT takes a bit more advantage of this (for instance, a bonus for castling is done in pawn eval, rather than in eval).

# 6 Evaluation

Both IPPOLIT and Rybka have evaluation functions ☺. Rybka has two (split as white/black), though except at the very end, I think they do exactly the same thing. However, it is still notable that while IPPOLIT follows the white/black split of Rybka at all other points, it does not do so here. A more detailed analysis of the size of positional components is given in the appendix.

## 6.1 Evaluation, first steps

The first step for each is to use the material token to get the material evaluation. As noted above, IPPOLIT has a overflow check for promoted pieces, and computes the material directly in this case. In other cases, it gets the exact value from the table. Rybka gets the material *adjustment* from the table, and seems simply to hope that this will not be problematic when there is an overflow condition (like 2 queens for one side).

## 6.2 Evaluation: pawn endgames

The next step is to see if the position is in evalhash, and then both run special code in the case of a pawn endgame. Both use a similar valuation involving the frontmost free pawn, but with different values, and (notably) IPPOLIT checks if the opponent has a non-passed pawn that might be even further advanced. IPPOLIT then has a `KP` vs `K` bitbase (two 120K files in the source), and a check for draws with pawns only on the `a`-file or `h`-file (lacking in Rybka). Some of the details of array access (or indexing) are a bit different also (for whether the king is in the square of a pawn, depending on who is to move).

## 6.3 Evaluation: lazy conditions

The condition for lazy evaluation is a bit different (even given the bounds as passed variables), as Rybka varies the bounds based upon whether a move is a capture and whether the previous move was lazy, while IPPOLIT only looks at how many previous moves were lazy (Rybka does not keep a running total). Rybka varies some "positional gain" parameters when lazy is triggered, but IPPOLIT does not. I'm not exactly sure what these do in Rybka in any case. Later versions of IPPOLIT/RobboLito seem to have eliminated lazy eval in various endgames, but I think the original used it in this case. Of course, both call an `evaluate_mobility` function when the lazy condition holds.

## 6.4 Mobility evaluation (special function)

Both compute mobility of all pieces, and determine attacked squares, and if in-check. The order in Rybka is `PNBRQKpnbrqk`, while the order in IPPOLIT is `KkNBRQnbrqPp`. IPPOLIT also has to keep track of xrays.

## 6.5 Evaluation of pawns

Pawns are evaluated in a separate functions, which I describe below.

## 6.6 Evaluation of pieces

Throughout the evaluation of pieces, Rybka and IPPOLIT both keep running totals for squares that are attacked, but IPPOLIT also keeps track of pieces that give check, and pin/xrays. Both keep track of "safe" squares for mobility, though its use is not always the same. The "main" evaluation tools of each are: mobility, pieces that are attacked (taking loose pieces and good-SEE attacks into account), and guarding one's own king. Both use these a lot.

### 6.6.1 Evaluation of pieces: pawn mobility and queens

Both Rybka and IPPOLIT look at pawn mobility first (with respect to non-pawns), though Rybka counts it differently on the side of the board of the opposing king. The next consideration is queens, with mobility (and xrays) being computed, and a score added for safe mobility (with Rybka adding a bit for "forward mobility"). This is done in conjunction with whether the queen gives check, or attacks a square next to the opposing king, or guards her own king. Loose pieces (not guarded by a enemy pawn) are added, then a subtraction if an enemy pawn attacks the queen. Finally a 7th rank bonus is added. Other than mobility difference (and the differences forced by xray computation) this is about the only place that Rybka and IPPOLIT really diverge for queen evaluation (though of course the numeric values differ throughout). Both demand that the opposing king or an enemy pawn be on the 7th/8th rank for a 7th-rank bonus, but IPPOLIT gives a "doubled on the 7th" bonus only if the opposing king is on the 8th, and the queen actually guards the rook that it also on the 7th.

### 6.6.2 Evaluation of pieces: Rooks

Rybka next does bishops, while IPPOLIT turns to rooks. I will look at rooks. After the usual mobility, xray, king attack/guards, IPPOLIT looks for attacks of loose enemy pawns and minor pieces, then if the rook attacks an enemy queen, and then if the rook is attacked by an enemy pawn (this is all part of a building-up routine, in which all good-SEE attacks will be noted). Rybka does some funky stuff with mobility and pawn skeletons, but also essentially does all the above, with a few extra additions (for instance, do two rooks guard each other when the opponent has a queen?). The usual additions for open files are then done – each has a score for an open file that is blocked by a fixed opposing minor piece, though the computation of this condition is different (mask versus bit-scan). IPPOLIT has a bonus for a rook outpost, which has a different notion in Rybka (see below). Both then check if the rook is on the 8th, and IPPOLIT then just checks if the opposing king is also, while Rybka considers doubled major pieces on the 8th. Then the rook is checked for being on the 7th rank, with the doubling criteria differing as mentioned in the queens section. Finally, both check if the rook is on the 6th rank, again with side conditions about the opposing king and pawns.

### 6.6.3 Evaluation of pieces: Bishops

Rybka computes a pawn skeleton in pawn evaluation, and uses this with rooks and bishops. For use a "colour count" with pawns as one facet in detecting bad bishops, but the array is not the same, and it gets used quite differently. (Rybka uses it at the end of the evaluation routine, while IPPOLIT uses it directly here in the bishop evaluation. I think Rybka also uses it with drawishness).

Both consider mobility (safe square, and possibly forward-mobility being advantaged) and attacks and guards as with other pieces. Rybka again has a few extra evaluation items (such as pawn directly in front of a bishop), while IPPOLIT again counts outposts. The notion of a "outpost" in Rybka seems be simply that a pawn guards a minor piece (with no condition on the opponent's pawns), while IPPOLIT takes much more into account. The mechanism for trapped bishops is expanded with IPPOLIT to include a guarding (by a pawn) of the trapping square. The main notable feature of IPPOLIT here is the calculation of good/bad bishops based on the pawn skeleton, which is quite different from Rybka (which also does it later).

### 6.6.4 Evaluation of pieces: Knights

Rybka uses a quite specific notion of mobility with knights, while IPPOLIT just does safe forward mobility. IPPOLIT also does outposts. Both do the usual with attacked pieces. Rybka has a couple of extra evaluation items (like: knight is 2h+2v away from opposing king in an endgame). Rybka has a slight knight outpost score, while IPPOLIT has a barrage of bonuses beyond a simple outpost here (if the knight is guarded by a pawn, if it attacks an enemy, if it is centralised).

## 6.7 Evaluation: continued

Both now give an addition if a king attacks an unguarded pawn, though IPPOLIT does some of this in pawn evaluation. IPPOLIT then counts trapped rooks via a much different formula (and method), while Rybka does this after good/bad bishops. Both then handle king safety, but with a different condition and mixing (the end results can differ quite notably – see the appendix). Both then gives a bonus if a side has more than one good-SEE attack. IPPOLIT adds a bonus for a rook/queen that corrals a king on an edge file in the endgame, which is absent in Rybka.

## 6.8 Evaluation: passed pawns

Both turn to passed pawns, where the conditions are not exactly the same, though both have special considerations for rook endgames (and queen endgames), and the main points are: whether the pawn can move, whether it has a clear path, and attacks (of either side) to the squares in front of the pawn, with Rybka adding a couple of other points, though there are also some look-ups to tables that are all zero here.

## 6.9 Evaluation: sundries

Rybka next does the computation of good/bad bishops, using the pawn skeleton. This is noticeably different from IPPOLIT, though I'm sure not exactly what it does in all facets. Rybka then does trapped rooks (done above in IPPOLIT), and a bonus for castling, which IPPOLIT has in pawneval. Rybka has a penalty for a king having no fleeing square (absent in IPPOLIT).

## 6.10 Evaluation: finale

Each obtains the final score via a linear interpolation of opening/ending scores (as in Fruit), though their computations of "phase" are different. This is then squashed by a "drawishness" parameter (from pawneval) that again differs [Rybka has an asymmetry with black/white eval at this juncture: although $\min(x, 150) = -\max(-x, -150)$, the right shift by 8 (after re-scaling from drawishness) in conjunction with this is not the same for positive/negative values]. As noted above, both pack two 16-bit scores into a 32-bit quantity throughout the evaluation routine, and need to unpack it to do the interpolation. The details are different, since Rybka has an additional `0x80008000` added into this.

IPPOLIT then checks for special endgames (including the bishop/knight mate code that is a large macro expansion, and blind bishops) and does a scaling with the 50-move rule. Finally the bookkeeping of positional scores and evaluation hash concludes the function.

# 7 Pawn evaluation

Both consider islands, holes, doubled, isolated, and backward pawns. They compute whether a pawn if backwards by a nearly identical method. Rybka also counts a few other things (central pawns, for instance), and has more to track with rotated bitboards of skeletons. Both keep a square-colour accounting of pawns, and whether such pawns are blockaded, but the system is different in the details. Both compute a distance from a king to pawns, though Rybka's method uses a complicated array that is absent in IPPOLIT. Curiously, both use 10000 as an upper bound for this distance (Rybka actually uses this later, while IPPOLIT does not — when there are no pawns, Rybka computes the distance to centre of the board, while IPPOLIT ignores it). Rybka has a *much* more involved method to determine candidate and passed pawns and their scaling (I don't claim to understand every detail). Both give a bonus for an outside passed pawn, though the indexing-access to this differs (squares versus files).

## 7.1 Drawishness with pawns

The second part of pawn evaluation is "drawishness" and the table used by Rybka is not very much like the method of IPPOLIT (which counts the number of pawns and whether they are opposed). This is detailed more in an appendix. Rybka also looks at central pawns, and their blocked nature, while IPPOLIT does not. (Larry Kaufman tells me that this is largely oriented toward anti-human play).

## 7.2 Pawns: Shelter

IPPOLIT adds in a few extras here that Rybka did in regular eval, such as a bonus for the right to castle. Pawn shelter is then computed. Rybka largely uses the 3x4 blocks that appear in Strelka, though the central files differ, and pawn storms with blocked centres and kings on opposing wings (and the like) are taken into account. The method in IPPOLIT is different in its execution, as it computes (bit-scans) the first pawn of three files near the king (not always the same files as in Rybka), and then adds the values for these. IPPOLIT penalises a storming pawn that is blockaded, and uses an interpolation method for shelter when castling is possible, both being absent in Rybka.

## 7.3 Pawns: Long diagonal

Finally, both do something with the long diagonal. This is probably a good example to show how one can have "kind of the same idea" (the same label "long diagonal" applies to each), but have it be much different at a number of levels. Rybka subtracts a penalty of 40, 20, 10 millipawns (in the opening) depending on whether the long diagonal of the king has: no pawns on it; has only enemy pawns; or has only friendly pawns. IPPOLIT has a penalty only depending on whether a friendly pawn is on the diagonal, and it depends on

the rank of the pawn, and the file of the king. For instance, for the a-file, the penalty is 0,2,4,6,8,10 centipawns, so a white king on a1 would incur a penalty of 2 centipawns if there is a white pawn on c3 but none on b2. The long diagonal for a given king square is obtained from an array with each, though (for instance) it seems that Rybka doesn't consider a5 to be on the e1 diagonal, as it wouldn't do anything for shelter (I guess), and IPPOLIT (for instance) puts g7 on the "long diagonal" for h6.

# 8 Search

Both use an aspirative alpha-beta search with lots of pruning and various extensions. IPPOLIT has functions for: root, pv, qsearch pv, CUT, ALL, exclusion, low depth, qsearch, and the last 6 all have a version for when in check, and are all split for white/black. Rybka combines CUT/ALL/exclusion to one search, but has an extra null-window search called only from pv nodes (and which can then re-call pv-search). The notion of "low depth" for IPPOLIT is less than 4 ply, and I think this is 3 ply in Rybka. Rybka passes more parameters to the various functions, and these seem to have to do with controlling extensions somehow. In an appendix, I give some pseudo-code to indicate the structure of the search functions.

## 8.1 Aspiration

The aspiration in Rybka goes by 20, then 40, 80, 160, 320, ... while in IPPOLIT it is 8, 12, 18, 27, 40, 60, 90, 135, ..., so that Rybka doubles, while IPPOLIT multilpies by 3/2 and starts with a more narrow window. The condition as to when to use aspiration is also different; IPPOLIT starts at depth 7, while Rybka at depth 6, and IPPOLIT uses it for all non-mate scores, while Rybka turns to a full-window when the score exceeds 500 in size. Both have some mechanism for making an "immediate" move when the root position has already been analysed sufficiently. With IPPOLIT it checks that the "best move" is at least 50 ahead of other moves (via exclusion search to a lower depth), while Rybka's method is similar, though not the same.

## 8.2 Root search

The "root" searches are (like almost everything it seems) similar in the algorithmic sense, but different in details. For instance, Rybka uses this null-window PV-esque function, which IPPOLIT lacks. Both have "easy" moves, though with a different criterion. Both have a condition for "best move is being challenged" (that is, a zero-window scout search failed to reject an alternative – Rybka first turns to the zero-window PV-esque search before a full-blown PV-search) that can lead to more time being allocated, and both have a "best move now looks bad flag", though the margin of 25 centipawns in IPPOLIT is not the same in Rybka. The initial sorting of root moves is not exactly the same either,

especially with captures. Rybka quits at $6 + 60 \cdot 2$ half-ply, while IPPOLIT can go up to 250 – the reason might be in hash tokens, as Rybka only has 7 bits for the depth.

## 8.3 PV nodes

The main PV searches follow the general alpha-beta mechanism. As noted above, Rybka has four extra parameters passed to this function (not counting the board pointer), which seem to control extensions. Both first do a hash lookup, though the condition (largely on depth) for when to use a hash entry at a PV node differs. The IID (iterative internal deepening) comes next: when there is no hash move and the depth is at least 6 (half-ply), IPPOLIT does a search at four ply less, then two ply less if this is successful. Rybka does the same, but the initial condition involves a larger depth, a different window is used with the IID (as opposed to IPPOLIT which adds/subtracts "depth" [in half-ply] from alpha and beta). Also, IPPOLIT does the same four ply, then two ply, in an alternative case, while Rybka reduces by only 1 ply when the depth is sufficiently small. Rybka also has a secondary "hash move" slot (for move ordering) if the reduced-by-4-ply search and reduced-by-2-ply search disagree.

### 8.3.1 PV nodes: extensions

Both then generate evasions when in check, with the details of ordering with trans/history/capture/other scores varying. Both give a "singular" extension of 2 half-ply for a forced move when in check, and a half-ply extension if exactly 2 legal moves exist (I didn't actually verify that both only generate legal moves in evasions, as interposing with pinned pieces can be tricky).

Both then test if the hash move is "dominant" with all other moves bad. This is done via the "exclusion" search, though the conditions with its use are quite different. Both first ensure that the depth is at least 16 half-ply (however, this "16" appears later with IPPOLIT, but varies with Rybka), and the hash move is vaild, and that we don't already have a 2-half-ply singular extension from a forced move (when in check). Rybka then determines if this hash move generates additional extensions (and so the whole shebang could be skipped), while IPPOLIT just computes its value with depth reduced by 10 half-ply, this being 12 half-ply with Rybka. The methods now diverge a lot, with Rybka doing something different when the depth is at least 26 half-ply, and keeping track of previous singular moves, while IPPOLIT has a much simpler accounting. Both call the "exclusion" search, though the scout value is different, as in IPPOLIT it varies with the depth. IPPOLIT also calls "exculsion" twice (first with the scout less by depth/2, then with it less by depth), adding one singular half-ply for each time. Rybka does not have this, though as I say, keeps track of singular moves in a different manner.

### 8.3.2 PV nodes: next-move loop

Both then turn to a `next_move` loop. Moves that allow a repetition are (pre-)rejected in a similar manner involving to/from square detection (when scout is bigger than some bound, and this bound differs at the various times this idea is used, at least in Rybka), though the exact criterion differs. Extensions are then added, starting with passed pawn pushes (though the definition/calculation of these differs), and also whether a move gives check. When a move fails high, Rybka can first use the null-window PV-esque search (before full-blown PV), which is nonexistent in IPPOLIT. After the move is un-done, Rybka has some hashing code that I don't see in IPPOLIT. Both then take care of history, etc., with the necessary details differing. As noted above, Rybka has 64-bit hash entries, with only 10 bits for a score, and 2 history tables rather than just one.

Since IPPOLIT lacks a null-window PV-esque search, I will skip it.

## 8.4 qsearch for PV nodes

Both have a PV qsearch that gets called from a PV node when the depth gets below one ply. The delta prunings follow a similar pattern (pawns, minor, rooks, with a skipping if no pawn is attacked), though IPPOLIT has fixed margins while Rybka's depend on the pre-evaluation (game phase). Both then do capture/promotiion generation, but IPPOLIT notes the hash move before entering the `next_move` loop, while Rybka does it during the sort phase of this loop. Both track bad captures, but IPPOLIT actually looks at them later, while Rybka just seems to do something with them in `quiet_checks`. Rybka uses a lazy margin (when calling evaluation) throughout this search, while IPPOLIT does not.

### 8.4.1 qsearch for PV nodes: bad captures and quiet checks

After the main loop, IPPOLIT then does bad caps if the depth is positive (necessarily 1 half-ply), and then both look at quiet checks. The condition to look at these is different, both in the depth required, and in the margin from alpha/beta to evaluation (IPPOLIT has this margin depend on depth, while Rybka does also, but modifies it via the pre-evaluation). Both then use their "positional gain" search(es), with again the condition for doing this differing both in depth required and margin of alpha/beta to evaluation.

### 8.4.2 qsearch for PV nodes: when in check

Both have a separate function for when in check at a pv qsearch node. Again both use hash, though I don't think Rybka tracks the hash move. Both have a delta-pruning of pawns then minors, with different margins, though here Rybka's are fixed (not dependent on pre-evaluation as in other places). IPPOLIT puts ordering-scores on the moves in the generation, while Rybka does it afterwards, largely only caring about captures.

In the ordering, Rybka decrements the depth by 1 (it matters little whether it is a half-ply or a whole at this point), with the ordering being skipped if there is zero or one moves (and so the depth remains the same). IPPOLIT does similar, but has the "curious" condition that the number of moves should not be *exactly* one for its decrementing of the depth. However, I don't see how this could come from Rybka, and IPPOLIT uses the more standard "number of moves is greater than one" condition in normal qsearch.

After this ordering by Rybka, both loop over moves, etc. IPPOLIT seems to keep track of moves that exceed alpha, and hash them directly with a lower bound (and the same in normal PV search), while Rybka does not.

## 8.5 CUT/ALL/exclude nodes

These form one function in Rybka, and three functions in IPPOLIT, though with a variant for white/black, and for being in check, brings it to a total of 4 or 12 functions respectively. As with PV nodes, Rybka has an extra four parameters in the function calls. I think Rybka doesn't look at the 50-move rule here, but IPPOLIT does. Both first look at hash, with a condition for lower/upper bounds combined with whether a node is CUT/ALL. In Rybka, the "exclude" search has a hashkey that is varied from the typical via the location of the opposing king with the to/from of the excluded move. In IPPOLIT, it uses the `to` for the opposing king and the `from` with one's own king.

### 8.5.1 CUT/ALL/exclude nodes: Null move

After the hashing, the next item is null move. Both require (as Fruit) that the evaluation be greater than the scout value, while Rybka also has a condition with a hash depth. IPPOLIT turns off null move when a side has only a minor left, while Rybka waits until only pawns are left. If the depth is large enough, Rybka then first does an IID-like search at depth minus 12 half-ply, and if this fails low, then skips null move. This does not appear in IPPOLIT, and neither does Rybka's next step, which (again at large depth) is to first check the null move at a much smaller depth (maybe 6 ply) with a larger window. After this, Rybka then does the "normal" 3 ply reduction, while IPPOLIT has a standard of 3 ply, but adds more depending on the difference between the scout and evaluation (this does not appear in Rybka). After null move, if it was successful, both store the hash only if a hash move does not exist.

### 8.5.2 CUT/ALL/exclude nodes: IID and singular extensions

The next step is IID, which I think both call only at CUT nodes (not sure with Rybka), and Rybka has a few extra conditions on when to call it, and a different depth parameter (IPPOLIT requires 6 half-ply). Both use a vanilla 2-ply reduction. For CUT nodes there is next a "singularity" check. If a hash move exists, and the depth is 16 half-ply (in IPPOLIT, not the same in Rybka), then the exclusion search is called. As with PV nodes, IPPOLIT uses a varying

window based on the depth. It also has "supersingular" extensions (more than 1 ply) when the height of a node is sufficiently small compared to its depth. I'm not sure whether these could theoretically blow up the search, but they do allow forcing lines to be found more readily, I suspect. The "exclusion" call in Rybka differs in both parameters (scout value and depth reduction), and while Rybka does seem to have "supersingular" ideas, they seem much more limited than in IPPOLIT (and involve the four extra function parameters somehow).

#### 8.5.3 CUT/ALL/exclude nodes: Type of search

Both next turn to choosing a "type" of search. When the scout and evaluation differ by a lot (depth-variable in each, though via a formula in IPPOLIT and a table in Rybka), only captures and checks are considered. Both also can eliminate enemy pawns from the capture-target (a form of delta pruning) in this step.

#### 8.5.4 CUT/ALL/exclude nodes: next move loop

Both of the `next_move` loops both first check for a repetition possibility (as above), and then skip moves that look hopeless. The criterion here is similar (the count of moves is more than 5 [both use the same], the phase is "quiet moves", the move does not give check, either directly or via an xray), with the primary condition being that the evaluation and scout differ by a sufficiently large amount depending on the depth — however, Rybka uses a table for this amount, and IPPOLIT has a formula, with both varying for the type of node. I can also mention that Rybka has more phases in `next_move`, for instance, a secondary hash move (from IID), and more killers potentially. Rybka then has a second condition for this "move-skipping" that IPPOLIT does not do at CUT nodes. This one is similar, but involves whether a move is a bad-SEE quiet move. Again the details differ. (See the appendix for pseudo-code).

#### 8.5.5 CUT/ALL/exclude nodes: MakeMove and more pruning

Then `make_move` is called, and eval with a lazy margin (IPPOLIT has a fixed value, while Rybka gets it from pre-evaluation). Both then turn to extensions, with IPPOLIT extending passed pawn pushes (again calculated differently) in a slightly different manner, and also varying the rank-condition for such a push depending on whether the node is CUT or ALL. IPPOLIT also has a "piece-swap" extension at CUT nodes. If the move does not give check, there is then (in both) another opportunity to ignore the move (undo it w/o recursing), which depends on the post-move evaluation compared to the scout value — as usual, the details differ. Then late-move reduction is applied, with the values differing, for all types of nodes. Rybka starts out at 6 half-ply at a CUT node, and increases to 7, while IPPOLIT starts at 6, but has a logarithmic increase. For ALL nodes, both start at the third move (assuming that the `quiet_moves` phase is reached), with a reduction of 2 half-ply, which IPPOLIT increases logarithmically again, with Rybka eventually incrementing it to 3.

After the unmake move, Rybka again does some hash updating, and then both turn to history and such as before. Both return scout-minus-one when a move fails low.

#### 8.5.6 CUT/ALL/exclude nodes: evasion search

Next there is the evasion search at these nodes. Both use hash, then a singularity check via exclusion (with different parameters as above). Both manage not to generate moves before testing the hash move, though the mechanism seems different. Both use hash/killer/capture/history for ordering though the details differ. Both check for repetition, but then Rybka has a "skip this move" (without recursing) condition which IPPOLIT lacks (involving bad-SEE interposition — this condition also appears in the low depth evasion search). Both use LMR here, with different details again, and both extend a half-ply (for check) in the "early" part of the game, though the definition of this differs (as does the computation of the phase). Both extend by a half-ply (with a half-ply already being extended for the move that gives check) in the "early" part of the game, with the computation of this differing in them.

### 8.6 Nodes of low depth

As mentioned before, this is a separate search, for nodes of depth at least 1 ply, but less than 4 ply (IPPOLIT) or 3 ply (Rybka). There is no difference between CUT and ALL here in IPPOLIT. I think Rybka has a parameter for it, but I don't know how it is used.

#### 8.6.1 Nodes of low depth: first pruning

Both first check if the situation is "hopeless", which in IPPOLIT means the evaluation and scout differ by 1125 or more. Rybka uses a different value, and returns a different value in this case. Rybka then can immediately jump to qsearch if the depth is less than 2 ply, and the margin is large. I don't see this in IPPOLIT, at least before hash. Hash follows a typical pattern, and then another "give up" condtion is queried. This is again a margin between evaluation and scout, and in IPPOLIT is "70 plus 10 times the depth in half-ply", while Rybka uses values from pre-evaluation for the details. IPPOLIT then sets the "best value" to the minimum of scout-minus-one or evaluation, and Rybka does something similar, but not the same. Null move is then called, and there are fewer conditions, as one always recurses into qsearch. Rybka even does null move in pawn endings here, while IPPOLIT still requires more than a minor piece.

#### 8.6.2 Nodes of low depth: type of search

As with the CUT/ALL nodes, the next step is to choose a "type" of search. IPPOLIT has three types, while Rybka only has two. The common "secondary" type of search is as before, with a large difference between evaluation and scout

leading to only captures and checks being considered. However, IPPOLIT has a much more thorough delta pruning here, which is mostly absent in Rybka. The third "type" of search in IPPOLIT (absent herein in Rybka) is when the depth is less than 2 ply and the evaluation is bad but not horrible, and then "positional gain" moves are also considered in addition to captures and checks. Rybka only considers "positional gain" moves in qsearch.

### 8.6.3 Nodes of low depth: NextMove loop

In the `next_move` loop, both (pre)eliminate possible repetitions, though Rybka uses a different value for the scout comparison. Next both have two possible ways of skipping a move. The first involves: the move phase being quiet moves, the count of moves being large (different details), the move not being check, having a piece (IPPOLIT only), and finally a comparison involving evaluation, scout, and "positional gain", with Rybka using a bound from pre-evaluation, and IPPOLIT a value that depends on the depth and the move count. The second involves bad-SEE moves (including captures, depending on the depth), with the usual similarities and differences.

### 8.6.4 Nodes of low depth: MakeMove and beyond

After the move is made (and if it is not check), there is again the chance for "move skipping" in both, with the details of the condition differing. Both then simply reduce the depth by a ply (no extensions/reductions) and recurse. Both reduce by only a half-ply when a move gives check (this is true at all stages, but is subject to an additional half-ply extension in some places).

IPPOLIT does a checking that there is a legal move here (moves that are generated but ignored are assumed "legal"), and if not, returns a draw score (the same is done at CUT/ALL nodes). Rybka does not seem to do this. Also, IPPOLIT returns the "best value" upon fail low, while Rybka does an interpolation with the scout.

### 8.6.5 Nodes of low depth: evasion variant

In the evasion version of this function, both use hash, then have a mechanism for avoiding move generation until after the hash move is tried. The ordering of moves is as previously. Again Rybka uses a different value when checking for possible repetitions. Rybka seems to ignore many more moves here, while IPPOLIT only skips non-hash interposing moves that are bad-SEE when the scout value is not a being-mated score. Rybka (among its conditions) has something that is sort of like this, but rather different. Both reduce by a half-ply, except in the "early" game. IPPOLIT changes the `best_value` to scout-minus-one when a move was skipped and the `best_value` is a being-mated score, while Rybka does not.

### 8.7 qsearch nodes

Rybka has a couple of extra parameters here for passing arrays of attackers around it seems (for use with the second SEE function?). IPPOLIT does not have these. Both look at hash, with Rybka updating these extra parameters, and then both set `best_value` to something a bit bigger than the evaluation. The difference is fixed (5) in IPPOLIT, and varies with pre-evaluation in Rybka (essentially the value of a tempo is phase-dependent). If this `best_value` exceeds scout, the qsearch is ignored (stand pat). Both then do delta pruning for pawns, minor, and rooks, with the margins varying in Rybka via pre-evaluation. Both then generate captures/promotions, though I think Rybka avoids all underpromotions here (via a flag to capture-generation), except knight checks. IPPOLIT then tags the hash move in this move list, while Rybka does that in the move loop. Both have a condition regarding SEE, though Rybka will alternatively use this "secondary" SEE depending on how close evaluation and scout are. IPPOLIT has only bad captures and quiet checks in this function, while Rybka also includes a "positional gain" generator. The conditions for using quiet checks are also different, and Rybka has a bunch of bookkeeping things with skipping moves and repetition checks (not apparent in IPPOLIT). At the end of the function, Rybka does an interpolation of evaluation and scout for a fail-low move, and sets the depth of the hash entry in a way depending on whether "positional gain" search was attempted. IPPOLIT ignores all this.

### 8.8 qsearch nodes when in check

The evasion versions here have various differences, as IPPOLIT uses the hash move, and doesn't check if a move will lead to a repetition. Rybka also uses a separate evasion generator, rather than the usual one. Both do use delta pruning for pawns and minor (but not rooks), with the methodological differences of before. Both prune bad-SEE moves, though IPPOLIT has a condition with interposition. Both prune "hopeless" non-capture moves (using positional gain), though IPPOLIT requires the side-to-move to have more than a minor piece (null move condition) and the scout not to be a being-mated score. As with low depth nodes, IPPOLIT sets `best_value` to scout-minus-one when a move was skipped and the `best_value` is otherwise being-mated. Rybka has a few extra details with bookkeeping for the attack arrays.

## 9 Other

Such things as time usage and UCI/FEN parsing are unalike. Also, I know of no "bugs" in Rybka that IPPOLIT reproduces. For instance, IPPOLIT handles 2-rep versus 3-rep (at the root) more satisfactorily than Rybka.

# A Closeness of evaluation numerology

In this appendix, I give some idea of "numerical coincidences" that appear in evaluation. This is nontrivial to do, due to partial re-scaling, and also that one really needs *all* the numbers to reach a complete conclusion, as else it is easy to over-emphasize one genre or the other. However, I try to give a representative sample.

## A.1 Static Values

Here I choose king values, pawn values, and knight values. There are the actual PST values, and also general piece sacling. IPPOLIT visibly uses a Fruit-like formula to build PST values, while Larry Kaufman indicates that (aside from pawns) a similar method was used a first approximation with Rybka, and then adjusted in some cases.

### A.1.1 Kings

Here are the PST values for kings in the opening, Rybka on the left (scaled to centipawns, though the numbers are millipawns internally), and IPPOLIT on the right.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 23 | 29 | 1 | -15 | -15 | 1 | 29 | 23 |
| 26 | 32 | 4 | -12 | -12 | 4 | 32 | 26 |
| 27 | 33 | 5 | -11 | -11 | 5 | 33 | 27 |
| 31 | 37 | 9 | -7 | -7 | 9 | 37 | 31 |
| 34 | 40 | 12 | -4 | -4 | 12 | 40 | 34 |
| 36 | 42 | 14 | -2 | -2 | 14 | 42 | 36 |
| 41 | 47 | 19 | 2 | 2 | 19 | 47 | 41 |
| 43 | 49 | 20 | -1 | -1 | 20 | 49 | 43 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 10 | -20 | -40 | -40 | -20 | 10 | 5 |
| 15 | 20 | -10 | -30 | -30 | -10 | 20 | 15 |
| 25 | 30 | 0 | -20 | -20 | 0 | 30 | 25 |
| 30 | 35 | 5 | -15 | -15 | 5 | 35 | 30 |
| 35 | 40 | 10 | -10 | -10 | 10 | 40 | 35 |
| 38 | 43 | 13 | -7 | -7 | 13 | 43 | 38 |
| 41 | 46 | 16 | -4 | -4 | 16 | 46 | 41 |
| 44 | 49 | 19 | -1 | -1 | 19 | 49 | 44 |

Here the IPPOLIT values come from $\mathrm{R}[rank] + \mathrm{F}[file]$ where

$$\mathrm{R} = [4, 1, -2, -5, -10, -15, -25, -35] \quad \text{and} \quad \mathrm{F} = [40, 45, 15, -5, -5, 15, 45, 40].$$

As can be noted, there is strong correlation, at least for the first few ranks.

Here are the endgame values for kings.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -55 | -32 | -19 | -9 | -9 | -19 | -32 | -55 |
| -35 | -12 | 1 | 11 | 11 | 1 | -12 | -35 |
| -23 | 0 | 13 | 23 | 23 | 13 | 0 | -23 |
| -18 | 5 | 18 | 28 | 28 | 18 | 5 | -18 |
| -22 | 1 | 14 | 24 | 24 | 14 | 1 | -22 |
| -35 | -12 | 1 | 11 | 11 | 1 | -12 | -35 |
| -53 | -30 | -17 | -7 | -7 | -17 | -30 | -53 |
| -76 | -53 | -40 | -30 | -30 | -40 | -53 | -76 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -53 | -30 | -14 | -8 | -7 | -14 | -30 | -53 |
| -35 | -10 | 2 | 8 | 8 | 2 | -10 | -35 |
| -24 | -3 | 12 | 18 | 18 | 12 | -3 | -24 |
| -18 | 3 | 18 | 27 | 27 | 18 | 3 | -18 |
| -23 | -2 | 13 | 22 | 22 | 13 | -2 | -23 |
| -29 | -8 | 7 | 13 | 13 | 7 | -8 | -29 |
| -40 | -15 | -3 | 3 | 3 | -3 | -15 | -40 |
| -73 | -50 | -34 | -28 | -28 | -34 | -50 | -73 |

Again the IPPOLIT values come visibly from a formula involving centralisation, and as Rybka presumably used a similar Fruit-like methodology at one stage, the "pattern" similarity is almost irrelevant compared to any numerology. Unfortunately, I don't really see any great statistical test to apply here. For instance, there are many similarities, but then `b2/g2` differ by 15 centipawns (more generally, the 2nd-rank weightngs differ rather notably).

### A.1.2 Knights PST

Here are the raw PST values for Rybka (opening on left, ending on right), and then for IPPOLIT, again after re-scaling Rybka to centipawns.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -127 | -40 | -30 | -25 | -25 | -30 | -40 | -127 |
| -35 | -20 | -10 | -5 | -5 | -10 | -20 | -35 |
| -25 | -10 | 0 | 5 | 5 | 0 | -10 | -25 |
| -25 | -10 | 0 | 5 | 5 | 0 | -10 | -25 |
| -30 | -15 | -5 | 0 | 0 | -5 | -15 | -30 |
| -40 | -25 | -15 | -10 | -10 | -15 | -25 | -40 |
| -55 | -40 | -30 | -25 | -25 | -30 | -40 | -55 |
| -75 | -60 | -50 | -45 | -45 | -50 | -60 | -75 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -25 | -19 | -15 | -13 | -13 | -15 | -19 | -25 |
| -17 | -11 | -7 | -5 | -5 | -7 | -11 | -17 |
| -13 | -7 | -3 | -1 | -1 | -3 | -7 | -13 |
| -13 | -7 | -3 | -1 | -1 | -3 | -7 | -13 |
| -15 | -9 | -5 | -3 | -3 | -5 | -9 | -15 |
| -19 | -13 | -9 | -7 | -7 | -9 | -13 | -19 |
| -25 | -19 | -15 | -13 | -13 | -15 | -19 | -25 |
| -33 | -27 | -23 | -21 | -21 | -23 | -27 | -33 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -120 | -21 | -10 | -6 | -6 | -10 | -21 | -120 |
| -16 | 0 | 11 | 15 | 15 | 11 | 0 | -16 |
| -7 | 9 | 20 | 24 | 24 | 20 | 9 | -7 |
| -5 | 11 | 22 | 26 | 26 | 22 | 11 | -5 |
| -11 | 5 | 16 | 20 | 20 | 16 | 5 | -11 |
| -20 | -4 | 7 | 11 | 11 | 7 | -4 | -20 |
| -36 | -20 | -9 | -5 | -5 | -9 | -20 | -36 |
| -58 | -42 | -31 | -27 | -27 | -31 | -42 | -58 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -15 | -10 | -5 | -2 | -2 | -5 | -10 | -15 |
| -8 | -1 | 3 | 5 | 5 | 3 | -1 | -8 |
| -3 | 3 | 8 | 10 | 10 | 8 | 3 | -3 |
| -4 | 1 | 6 | 10 | 10 | 6 | 1 | -4 |
| -6 | -1 | 4 | 8 | 8 | 4 | -1 | -6 |
| -10 | -4 | 1 | 3 | 3 | 1 | -4 | -10 |
| -15 | -8 | -4 | -2 | -2 | -4 | -8 | -15 |
| -22 | -17 | -12 | -9 | -9 | -12 | -17 | -22 |

These do not look too close, but upon adding 20 to the opening values of Rybka, and 10 to the ending, a closer congruence can be noted. Here are the opening and ending values for Rybka after this is done.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -107 | -20 | -10 | -5 | -5 | -10 | -20 | -107 |
| -15 | 0 | 10 | 15 | 15 | 10 | 20 | 35 |
| -5 | 10 | 20 | 25 | 25 | 20 | 10 | -5 |
| -5 | 10 | 20 | 25 | 25 | 20 | 10 | -5 |
| -10 | 5 | 15 | 20 | 20 | 15 | 5 | -10 |
| -20 | -5 | 5 | 10 | 10 | 5 | -5 | -20 |
| -35 | -20 | -10 | -5 | -5 | -10 | -20 | -35 |
| -55 | -40 | -30 | -25 | -25 | -30 | -40 | -55 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -15 | -9 | -5 | -3 | -3 | -5 | -9 | -15 |
| -7 | -1 | 3 | 5 | 5 | 3 | -1 | -7 |
| -3 | 3 | 7 | 9 | 9 | 7 | 3 | -3 |
| -3 | 3 | 7 | 9 | 9 | 7 | 3 | -3 |
| -5 | 1 | 5 | 7 | 7 | 5 | 1 | -5 |
| -9 | -3 | 1 | 3 | 3 | 1 | -3 | -9 |
| -15 | -9 | -5 | -3 | -3 | -5 | -9 | -15 |
| -23 | -17 | -13 | -11 | -11 | -13 | -17 | -23 |

In particular, the ending values now have a maximal difference of 2, and similarly (away from corners) with the opening values. Of course, one must

also account for the fact that both employ a Fruit-like method to generate these values, and that these additions of 10 and 20 to the Rybka values are not precisely recovered from the base piece values.

### A.1.3 Base values

Indeed, for the "base value" of knights, Rybka gets these from a 3-phase interpolation, while IPPOLIT has a 4-stage interpolation. In fact, Rybka's "3-stage" interpolation only really depends on two parameters (indeed, Larry Kaufman says that he demonstrated to Rajlich that the extra complication provided little reward).

The IPPOLIT interpolation is with $[265, 280, 320, 355]$, with the first two being used in the first 25% of the game, the middle two in the middle 50%, and the last two at the end. Rybka uses $[277, 318, 359]$ Also, Rybka uses a Fruit-phase of 1/2/4 for minor/rook/queen, while IPPOLIT uses 1/3/6.

For pawns, the IPPOLIT interpolators are $[80, 90, 110, 125]$, while these are $[78, 100, 122]$ in Rybka.

The Knight/Pawn adjustment in IPPOLIT is $[0, 2, 4, 5]$ (per pawn and knight), while in Rybka it is just $[3, 3, 3]$. Both have a Rook/Pawn adjustment, but Rybka has a Queen/Pawn adjustment while IPPOLIT does not.

### A.1.4 Pawns

The Rybka values for PST with pawns are no longer multiples of 10 when expressed in millipawns, and do not follow any readily discernible structure that I can see. IPPOLIT again has a Fruit-like method.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -175 | -50 | 19 | 88 | 88 | 19 | -50 | -175 |
| -195 | -70 | -1 | 68 | 68 | -1 | -70 | -195 |
| -205 | -82 | -13 | 56 | 56 | -13 | -82 | -205 |
| -213 | -90 | -25 | 40 | 40 | -25 | -90 | -213 |
| -219 | -97 | -37 | 23 | 23 | -37 | -97 | -219 |
| -224 | -103 | -46 | 11 | 11 | -46 | -103 | -224 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -16 | -16 | -44 | -72 | -72 | -44 | -16 | -16 |
| -36 | -36 | -64 | -92 | -92 | -64 | -36 | -36 |
| -50 | -50 | -78 | -106 | -106 | -78 | -50 | -50 |
| -62 | -62 | -90 | -118 | -118 | -90 | -62 | -62 |
| -66 | -66 | -94 | -122 | -122 | -94 | -66 | -62 |
| -66 | -66 | -94 | -122 | -122 | -94 | -66 | -62 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -170 | -50 | 10 | 80 | 80 | 10 | -50 | -170 |
| -180 | -60 | 0 | 70 | 70 | 0 | -60 | -180 |
| -190 | -70 | -10 | 60 | 60 | -10 | -70 | -190 |
| -210 | -90 | -30 | 40 | 40 | -30 | -90 | -210 |
| -220 | -100 | -40 | 30 | 30 | -40 | -100 | -220 |
| -230 | -110 | -50 | 20 | 20 | -50 | -110 | -230 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -20 | -40 | -60 | -80 | -80 | -60 | -40 | -20 |
| -40 | -60 | -80 | -100 | -100 | -80 | -60 | -40 |
| -50 | -70 | -90 | -110 | -110 | -90 | -70 | -50 |
| -60 | -80 | -100 | -120 | -120 | -100 | -80 | -60 |
| -70 | -90 | -110 | -130 | -130 | -110 | -90 | -70 |
| -70 | -90 | -110 | -130 | -130 | -110 | -90 | -70 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

The first two tables are for opening and endgame in Rybka, and the second are for IPPOLIT (re-scaled to millipawns). Again the numerology looks quite similar, though the slight difference in base value should also be included.

## A.2 Some components in evaluation

### A.2.1 Rooks

The first bonus occurs when the rook is on same line as opposing king with one unit intervening. The bonus depends upon the unit (either a pin or an x-ray). In IPPOLIT it is $(30, 50)$ millipawns (I re-scaled to these for convenience) for a friendly king or minor, and is $(20, 50)$ for enemy units, except for rooks (zero) and queens where it is $(100, 200)$. In Rybka, a friendly non-major (including pawns) is $(10, 20)$ millipawns, while an opposing pawn/minor is $(10, 20)$, an opposing rook is $(10, 10)$, and an opposing queen is $(100, 150)$.

The mobility bonus in Rybka is rather complicated, involving pawn skeletons. There is a bonus of $(6, 6)$ millipawns for each square of horizontal rook mobility ignoring pieces, and $(10, 11)$ millipawns (per square) for such vertical mobility. There is another $(25, 25)$ per square for general mobility, and a final $(5, 5)$ for mobility per square on ranks 5-8. IPPOLIT simply has $(20, 30)$ per square of rook mobility, and is thus somewhat lower. In all cases, mobility restricts to squares not attacked by an enemy pawn.

Other bonuses are more formulaic. Here is a table:

| Feature Name | Rybka | IPPOLIT |
|---|---|---|
| RguardK | (30,0) | (30,10) |
| RguardR (opp has Q) | (30,30) | - |
| OutpostRook | - | (10,20) |
| PguardR/OutpostRookGuarded | (10,10) | (30,40) |
| RattP | (30,60) | (20,30) |
| RattBN | (30,60) | (40,50) |
| RattQ | (60,100) | (50,50) |
| (opposing) PattR | -(60,100) | -(70,100) |
| RookHalfOpen | (10,70) | (30,60) |
| RookOpenFile | (185,70) | (200,100) |
| RookOpenFixedMinor | (105,8) | (100,0) |
| RookOpenMinor | (155,45) | (150,50) |
| RookHalfOpenPawn | (35,55) | (50,50) |
| RookHalfOpenKing | (163,0) | (150,0) |
| RookKing8th | (50,100) | (50,100) |
| DoubRook8thKingPawn | (10,20) | - |
| Rook7thKingPawn | (80,320) | (100,300) |
| DoubRook7thKingPawn | (10,20) | (100,200) |
| Rook6thKingPawn | (40,160) | (50,150) |

I have followed the IvanHoe naming conventions, though I might stress that these don't always mean exactly the same thing for the two programmes (such as doubled rooks on the 7th, which really means doubled majors in the first place, but in IPPOLIT they must guard each other), and the naming is not always that great (for instance, `RattP` means an pawn not guarded by another pawn). Note that each bonus is gained at most once (for instance, attacking two opposing unguarded pawns is the same as one).

### A.2.2 King danger

Here I give a random sampling of 10 different configurations. The process in both is the same, to say that a piece "hits" the opposing king if a square around it is attacked. There is then a multiplier based upon how many Rybka has extra bonuses when a piece has an x-ray attack (from a friendly piece) against a square near the enemy king (example: wRa1, wNa2, bPa7, bKb8, so the white rook x-ray attacks the `a7` square), but I ignore these. Rybka demands that the opponent have a queen on the board to give a penalty, while IPPOLIT does not. All of these values are scaled with the phase. Both only count at most one pawn that attacks the king area. The IPPOLIT values are re-scaled to millipawns.

| Attackers | N | Q | BQ | RN | QP | QR | RBB | RNP | QBN | QRBP |
|---|---|---|---|---|---|---|---|---|---|---|
| Rybka | 0 | 0 | 223 | 400 | 124 | 281 | 711 | 801 | 932 | 1057 |
| IPPOLIT | 20 | 50 | 270 | 250 | 200 | 320 | 610 | 450 | 780 | 1600 |

Note that Rybka also has a deduction when the king has no flee square.

## A.3 Some components in pawn evaluation

### A.3.1 General

Rybka has a few more items with pawn scoring, such as allowing the value of an (adjective) pawn to vary on the square, while IPPOLIT usually only has it as dependent on the rank. For instance, doubled isolated pawns on a closed file are deducted at $(24, 32)$ in Rybka unless they are on the board's edge, when it is $(20, 26)$. Upon re-scaling to millipawns, IPPOLIT just has $(20, 40)$. To get an idea of how close the numerology is, here is another (random) example: Rybka deducts $(39, 52)$ for a backward pawn on a closed file, while it is $(50, 50)$ for IPPOLIT.

For rank-based constructs, such as passed pawn values, there is again some similarity in numerology. For instance, IPPOLIT has a raw PassedPawnValue of $(100, 100)$, $(200, 250)$, $(400, 500)$, $(600, 750)$ (in millipawns) for the 4th-7th ranks, while Rybka has $(80, 110)$, $(200, 260)$, $(380, 500)$, $(610, 800)$, and also $(20, 20)$ for both the 2nd-3rd ranks.

On top of any slight differences in numerology, one must also recall that the conditions for these to be applied are not always the same.

### A.3.2 Drawishness

IPPOLIT has a formulaic way of computing drawishness from opposing pawn structures. Rybka has a large table for each of the $2^8 = 256$ possibilities of occupied files (by one side), and any opposing nature of the pawns is ignored. The Rybka table seems to depend on the "span" of the pawns, that is, the distance from the leftmost to the rightmost pawn (of a given side). I give a table of 10 random configurations, to show how much these can differ. Rybka uses base 256, while IPPOLIT has base 64, so I scale the latter. Rybka's drawishness deduction is zero for a side with 5 or more pawns, while IPPOLIT's becomes zero at 6 or more.

| White | Black | wR | wI | bR | bI |
|---|---|---|---|---|---|
| cgh | cdgh | 54 | 72 | 30 | 40 |
| de | abfg | 100 | 64 | 20 | 16 |
| abe | acdg | 60 | 48 | 20 | 24 |
| bdef | abceh | 42 | 32 | 0 | 12 |
| fg | efgh | 100 | 96 | 54 | 32 |
| abgh | acdg | 10 | 32 | 20 | 32 |
| abcde | abcd | 0 | 20 | 54 | 48 |
| ac | bc | 94 | 80 | 100 | 80 |
| bdegh | cefh | 0 | 12 | 30 | 32 |
| adeg | bcfg | 20 | 24 | 30 | 24 |

Here `wR` indicates the drawishness scaling by Rybka for white, under the given pawn configurations. Here "100" means a scaling of 100/256, of the first 150 centipawns for Rybka (by now Rybka has divided by 10 in the evaluation code), and the first 100 centipawns for IPPOLIT.

### A.3.3 Shelter and storm

Rybka has numerous arrays for shelter and storm, depending on whether kings are on opposite sides of the board, and whether the centre is blocked. IPPOLIT has (in essence) one, though the capacity to have it depend on the file of the king is present. Both of them use 3-file arrays for pawns in front of the king, though the choice of which files to use will differ.

The numerology in some cases is close. For instance, the IPPOLIT arrays for shelter are $[0, 15, 40, 50, 55]$ depending on the rank of a sheltering pawn on a "centered" file (usually that on which the king resides). Rybka has a similar array $[0, 14, 40, 50, 60]$ in the case when the kings are on opposite sides and the centre is not blocked. Rybka (re-scaled to centipawns) has slightly differing arrays for similar situations, such as $[0, 11, 35, 45, 55]$ when the kings are on the same side of the board. When the centre is blocked, the numbers are reduced. The other two IPPOLIT arrays are $[0, 5, 15, 20, 25]$ and $[0, 10, 20, 25, 30]$, while in Rybka the most analogous arrays are either: $[0, 6, 25, 30, 35]$ and $[0, 11, 25, 30, 35]$;

or $[0, 3, 20, 25, 30]$ and $[0, 8, 20, 25, 30]$. In no case is there an exact correspondence. Similar comments apply to storming pawns. The method in Rybka to determine the "central" versus "edge" is a big lookup array, while in IPPOLIT pointer-switching is used.

# B Schematic drawing of search functions

I have chosen a random sampling of search functions for comparison. I avoided CUT/ALL/exclude nodes, as Rybka has one function, while IPPOLIT has three.

## B.1 PV node (IPPOLIT)

```
Ensure ALPHA and BETA are sane
If (DEPTH <= 1 HALF-PLY), then use PV-qsearch instead
Check 50-move rule and for repetition.
Look for hash entry
 Return only if not ANALYSING and an exact score with large depth exists
If no hash-move and DEPTH is not too small (at least 6 half-ply)
 If DEPTH is at least 10 half-ply, do IID with 8 half-ply reduction, if OK:
 Call IID with 4 half-ply reduction (and DEPTH-fiddled ALPHA/BETA)
Else if DEPTH is at least 10 half-ply and DEPTH exceeds hash-depth by enough
 Call IID with 8 half-ply reduction
 If OK, call IID with 4 half-ply reduction
If in-check, then generate moves and order them
 If one (quasi)legal move, extensions is set 2 half-ply
 If two (quasi)legal moves, extensions is set to 1 half-ply
If DEPTH is large (16 half-ply), and hash-move exists, and (extensions < 2)
 If hash-move is legal, get SCORE from search at reduced depth (5 ply)
 Do exclusion search at reduced depth (usually 6 ply)
 If SCORE beats EXCLUDE by enough, set extensions set to 1 half-ply
 If SCORE beats EXCLUDE by even more, extensions is set to 2 half-ply
NEXT-MOVE LOOP
 If (ALPHA > 0) and MOVE allows a repetition, then ignore MOVE
 If (extensions < 2)
  If MOVE is dangerous pawn push, then set extensions to 2 half-ply
  Else if MOVE is check, or is a capture, or phase is early and in-check
  or passed pawn is pushed, then extensions is set to 1 half-ply
 If MOVE is hash-move or NEW_DEPTH is 1 half-ply, call PV-node
 Else If NEW_DEPTH is small, then call low-depth, else call CUT-node
  If SCORE beats ALPHA then call PV-node
 If SCORE is lower than ALPHA, update HISTORY if move is not special
 Record MOVE is good, and update HASH
 If SCORE beats BETA, update HASH, return SCORE
If no moves exist, determine if mated or stalemate
Update history and hash for best move, and return score
```

## B.2 PV node (Rybka)

```
If (DEPTH <= 1 HALF-PLY), then use PV-qsearch instead
Check for repetition.
Look for hash entry
 If DEPTH is low enough, then return if a useful bound is known
If DEPTH is sufficiently larger than hash-depth
 If DEPTH is large (say 12 half-ply)
  Then try IID at a much lower depth and if the return value beats ALPHA,
   Then try IID at a slightly lower depth (4 half-ply)
 Else just do IID at a minimally lower depth (2 half-ply)
If ALPHA is a lot bigger than EVAL, then use secondary history table
If in-check, then generate moves and order them
 If one (quasi)legal move, extensions is set 2 half-ply
 If two (quasi)legal moves, extensions is set to 1 half-ply
If DEPTH is large (16 half-ply), and hash-move exists, and (extensions < 2)
 If hash-move is check, or is a capture, or phase is early and in-check
  or passed pawn is pushed, then extensions is set to 1 half-ply
 If hash-move is recapture and previous EVALs did not differ much,
  then extensions is set to 2 half-ply
 If dangerous passed pawn is pushed, extensions is set to 2 half-ply
 If move is legal, then determine SCORE via search at reduced depth (6 ply)
 If SCORE beats ALPHA
  If DEPTH is large, then do exclusion search at reduced depth (7 ply)
  Else do exclusion search at reduced depth (5 ply)
  If hash-move is singular, then extensions is set to 2 half-ply
  // NOTE: the singular margin is different in the 2 "DEPTH is large" cases
NEXT-MOVE LOOP
 If (ALPHA >= 0) and MOVE allows a repetition, then ignore MOVE
 If MOVE is hash-move and is a recapture with previous EVALs close,
  then extensions is set to 2 half-ply
 If dangerous passed pawn is pushed, extensions is set to 2 half-ply
 If MOVE is check, or is a capture, or phase is early and in-check
  or passed pawn is pushed, then extensions is set to 1 half-ply
 If MOVE is hash-move or NEW_DEPTH is 1 half-ply, call PV-node
 Else If NEW_DEPTH is small, then call low-depth, else call CUT-node
  If SCORE beats ALPHA
   If DEPTH is large and other conditions involving ALPHA and BETA closeness
    then call PV-esque null-window search
   If SCORE still beats ALPHA then call PV-node
 If DEPTH is large enough, and HASH conditions are met, return SCORE
 If SCORE is lower than ALPHA, update HISTORY if move is not special
 If SCORE beats BETA, then record it is good, update HASH, return SCORE
 If SCORE is between ALPHA and BETA, record MOVE is good, increase ALPHA
If no moves exist, determine if mated or stalemate
Update history and hash for best move, and return score
```

## B.3 Low-depth (not in-check)

### B.3.1 Rybka

```
If EVAL is much lower then SCOUT, then return SCOUT minus 101
if DEPTH < 2 ply and EVAL is somewhat lower than SCOUT, use qsearch instead
Check for repetition, then do hash lookup
If EVAL is bigger than SCOUT by enough, then return fail-high
If EVAL >= SCORE then try null move
If SCORE exceeds EVAL by 100 or more, then use secondary HISTORY tables
If EVAL is enough lower than SCORE
 Set SEARCH to only do captures and checks
 If EVAL is even more lower than SCORE, ignore pawn captures also
NEXT-MOVE LOOP
 If SCOUT is big enough and MOVE can lead to a repetition, ignore MOVE
 If move-count is big, and MOVE-phase is quiet moves and MOVE is not check
  If positional gain of MOVE is too low with EVAL and SCOUT, ignore MOVE
 If MOVE is not a capture and is not a king move, ep, or hash-move,
  and does not give check, and bad SEE, then ignore MOVE
 If MOVE is a capture and DEPTH is low
  If EVAL does not exceed SCORE by a lot and the captured piece is not a pawn
   and MOVE is not trivially good-SEE, and is not a king move, ep,
   or hash-move, and does not give check, and is bad SEE, then ignore MOVE
 Make move, evaluate, check legality
 If MOVE gives check, recurse at DEPTH minus 1 half-ply
 Else if move-count is big and pos-gain is too low with SCOUT/newEVAL, undo
 Else recurse at DEPTH minus 2 half-ply, either to low-depth or qsearch
 If SCORE beats SCOUT then update HISTORY (if applicable) and HASH, return
Update HISTORY, interpolate best_value and SCORE, return
```

### B.3.2 IPPOLIT (low depth, not in-check)

```
Ensure SCOUT is sane
If EVAL is much lower then SCOUT, then return SCOUT minus 1
Check for repetition and 50-move rule, then do hash lookup
If EVAL is bigger than SCOUT by enough, then return fail-high
If EVAL >= SCORE then try null move
If EVAL is enough lower than SCORE
 Set SEARCH to only do captures and checks
 If EVAL is even more lower than SCORE, ignore pawn captures also
  If EVAL is even more lower than SCORE, ignore minors, and then rooks
Else if EVAL is lower than SCORE, but not too much lower
 Set SEARCH to do captures, checks, and positional-gain moves
NEXT-MOVE LOOP
 If SCOUT > 0 and MOVE can lead to a repetition, ignore MOVE
 If move-count is big, and MOVE-phase is quiet moves and MOVE is not check,
  and I have a piece, and pos-gain of MOVE is low with EVAL/SCOUT, ignore MOVE
 If move is not a capture or (DEPTH is low and is not trivially good SEE)
  and is not a king move, ep, or hash-move, does not give check, and bad SEE
  then ignore the MOVE
 Make move, evaluate, check legality, and if move gives check in POS_GAIN phase
 If MOVE gives check, recurse at DEPTH minus 1 half-ply
 Else if move-count is big and pos-gain is too low with SCOUT/newEVAL, undo
 Else recurse at DEPTH minus 2 half-ply, either to low-depth or qsearch
 If SCORE beats SCOUT then update HISTORY (if applicable) and HASH, return
If move-count is zero and ordinary moves were considered, return DRAW
Update HISTORY and return best_value
```

## B.4 qsearch when not in-check

### B.4.1 IPPOLIT

```
Ensure SCOUT is sane, check repetition and 50-move rule, then do hash lookup
Set best_value to EVAL plus a tempo, return if this beats SCOUT
Do delta-pruning, first captures only, then ignore pawns, then minors, rooks
Generate captures, then ensure hash move will go first when ordering
NEXT-MOVE LOOP (CAPTURES)
 Find next move with highest ordering score
 If MOVE is not hash-move and not a discovered check, and is bad SEE, ignore
 Make move, evaluate, check legality
 Recurse: If SCORE beats SCOUT, then update HASH and return (fail-high)
If DEPTH is not too low and EVAL and SCOUT are sufficiently close
 Generate quiet checks
  For each: make, eval, check legal, recurse, if fail-high, update & return
Update HASH and return (fail-low)
```

### B.4.2 Rybka

```
Check for repetition then do hash lookup
Set best_value to EVAL plus a tempo, return if this beats SCOUT
Do delta-pruning, first captures only, then ignore pawns, then minors, rooks
Generate captures and LOOP:
 Find next move with highest ordering score
 If MOVE captures a king, return +CHECK_MATE
 If MOVE is not a discovered check
  If MOVE is not a promotion, nor ep, and SCOUT and EVAL differ by a lot
   If the capture square is not notable and badSEE2, ignore the MOVE
  Else if move is badSEE, ignore the move
 Increment move count(s), and if EVAL+(move-count) beats SCOUT, return
 Make move, evaluate, check legality
 If move gives check
  Determine how, update to-from squares, and delta-pruning
 Recurse to qsearch (either in-check or not, passing to-from to former)
 If recursion returns a good move, update to-from squares
 If SCORE beats SCOUT, then update HASH and return (fail-high)
If DEPTH is not too low and EVAL and SCOUT are sufficiently close
 Generate quiet checks and LOOP
  If SCOUT is big enough and MOVE can lead to a repetition, ignore MOVE
  Make move, evaluate, check legality, update to-from squares
  Recurse to qsearch in-check
  If SCORE beats SCOUT, then update HASH and to-from, return (fail-high)
If DEPTH is not too low and EVAL and SCOUT are sufficiently close
 Generate posgain moves (two functions, depending on SCOUT/EVAL), LOOP:
  If SCOUT is big enough and MOVE can lead to a repetition, ignore MOVE
  If MOVE is badSEE ignore it
```

```
  Make move, evaluate, check legality, ensure MOVE doesn't give check
  If MOVE does not gain enough, ignore it
  Recurse, and if SCORE beats SCOUT, update HASH/to-from, return (fail-high)
 Interpolate EVAL and SCOUT
Update HASH and return (fail-low)
```

## B.5 qsearch when in-check

### B.5.1 IPPOLIT

```
Ensure SCOUT is sane
Check for repetition and 50-move rule, then do hash lookup
Set best_value to mate score, depending on height
Do delta-pruning, first captures only, then ignoring pawns, then minors
Generate moves, if more than one move, then decrease DEPTH
Ensure hash move will go first when ordering
NEXT-MOVE LOOP
 Find next move with highest ordering score
 If MOVE is interpose and SCOUT is not a mate score,
  and MOVE is not the hash-move and SEE is bad, ignore MOVE
 If MOVE is not a capture, not a hash-move, and I have a piece,
  and posgain is not enough, and SCOUT is not a mate score, ignore MOVE
 Make move, evaluate, check legality
 Recurse to qsearch
 If SCORE beats SCOUT, then update HASH and return (fail-high)
If moves have been ignored and best_value is a mate score
 set best_value to SCOUT minus 1
Update HASH and return (fail-low)
```

### B.5.2 Rybka

```
Check for repetition then do hash lookup
Generate moves, if more than one move, then decrease DEPTH
// NOTE: a MASK is passed to the function that determines delta-pruning
NEXT-MOVE LOOP
 Find next move with highest ordering score
 If MOVE captures a king(?) return +CHECK_MATE
 If MOVE is not a capture and not a king move (interposition) and not ep
  and SEE is bad, ignore MOVE
 If SCOUT is big enough and MOVE can lead to a repetition, ignore MOVE
 If MOVE is not a capture (or promotion) and doesn't give check (I think?)
  and posgain (from a fixed piece with to-from?) is not enough, ignore MOVE
 Make move, evaluate, check legality
 Recurse to qsearch
 If SCORE beats SCOUT, then update HASH
  If MOVE is a capture, update MASK. Return (fail-high)
Update HASH and return //NOTE: function was passed a fail-low value to return
```