Daniele Guglielmo Gatti

COMPUTATIONAL MONASTICISM

Research Notes from a Homemade Large-Scale Selfmate Verification Project

Babson Task
(Selfmate in 11½)

1ª CATEGORIA NAZIONALE
1987
GARE AGONISTICHE

Oltre il calcolo, l'idea.
Oltre il rumore, il silenzio.
Oltre la massa, l'originalità.

STUDI
AUTOMATI
COMPOSIZIONE
BELLEZZA
PRECISIONE
VERITÀ

Selfmate Research
- Analysis
- Optimization
- Hash Strategy
- Human Insight
- Sustainable Compression

Goal: Verification
- Reduce
- Simplify
- Solve

From impossible to possible.
From brute-force to knowledge.

Creatività instancabile.
Bellezza senza tempo.

Q.E.D.

Non cerco approvazione.
Cerco solo la verità nascosta nelle posizioni.
DGG

ISOLATO PER SCELTA.
OSTINATO PER NATURA.
CREATIVO PER NECESSITÀ.

JULY 2026

Introduction

Some years ago, a selfmate chess problem was conceived around a very ambitious objective: creating a long forced structure whose correctness could survive complete brute-force verification. The theme was also ambitious: Valladao Task + AUW. At that time, the problem was never fully tested to the end, partly because of hardware limitations, partly because of the enormous analytical complexity involved. As a consequence, it remained in a sort of limbo: neither truly accepted nor truly refuted. Partial tests were highly encouraging, but they were not enough.

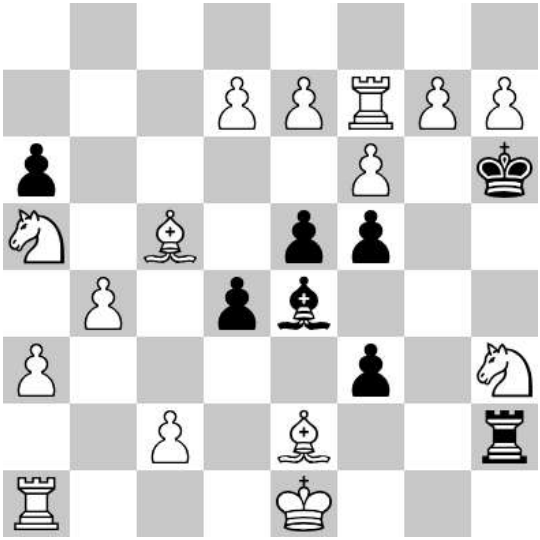
Eight years later, with inadequate machines, infinite patience, and a perhaps unhealthy amount of stubbornness (which won't be surprising for those who know me well), and after many failed attempts due to early discouragement, I finally managed to complete the verification process and reach the original goal.

The problem is certainly not perfect. But perfection was never the point here. The point was demonstrating that even highly experimental long selfmates can aspire to full soundness, provided one is willing to invest the necessary time and analytical effort.

If this book encourages even one composer to attempt an ambitious brute-force-tested S#13 (or more!) instead of

stopping at "probably correct", then its presence here will have served a purpose.

This is the "monster" we are evaluating:



Legality was proofed for a hair's breadth! Just one Black pawn capture and one White Queen missing, and the configuration of white pawns is ... "just reachable".

Try to put this position on Gustav, set the stipulation at S#13, and launch the analysis with the "Zugzahl Unbekannt" option activated. After some time, you will notice a discouraging fact: the required time growth increases at every successive depth with factors that could

reach x10, x15, x20, and the more you get deep into the analysis, the more you feel discouraged. Then you make the count, even if you know it's only an estimated time (then, the real necessary time could be even much longer) ... and you think "Ok, this is not feasible, it would require years or decades".

I tried to prove that this assumption is not true.

And this is my report.

TECHNICAL DATA

About the problem:

FEN: 8/3PPRPP/p4P1k/N1B1pp2/1P1pb3/P4p1N/2P1B2r/R3K3
Stipulation: **S#13**

Composed by: **Daniele Guglielmo Gatti**

Published on DIE SCHWALBE (2018) – No awards

Solution: **1. h8=Q+! Kg6 2. g8=R+! Kxf7 3. e8=B+! Ke6 4. d8=N+! Kd5 5. c4+! dxc3 en passant 6. O-O-O+! Bd3 7. Rxd3+ Ke4 8. Bf3+! Kxd3 9. Nf4+! exf4 10. Qxh2!!** [zugzwang] **c2 11. Qe2+!** [that's the deep strategic reason why 1. h8=R+? fails – no check possible!] **Kc3 12. Be3!** [zugzwang] **fxe3 13. Qd2+ exd2#** model mate.

About the analysis:

Beginning: **April 7th, 2026 – 6:00 A.M.**

Hardware used: CPU Intel i3-4150 3,50 GHz – 2 physical cores (4 threads) – internal memory L1 Cache 128 Kb; L2 Cache 512 Kb; L3 Cache 3 Mb - Overclocking NOT supported; 16 GB RAM DDR3 at frequency 1600 MHz. HDD system drive.

From 5th June, 2026 *until the end, an help came from an Intel i5-7400 3.00 GHz with HyperBoost (Auto-*

Overclock up to 3.50 GHz) and with 16 GB RAM DDR4 at frequency 2400 Mhz – 4 cores / 4 threads, with internal memory L1 cache 256 Kb; L2 cache 1 Mb; L3 cache 6 Mb. SSD system drive + HDD data drive.

Testing method: BRUTE-FORCE with Gustav 4.2h (developed by Olaf Jenkner).

Hashtables used: from 64 Mb to 13488 Mb for a single Gustav instance.

Analysis non-stop night and day, with a variable number of Gustavs (8 maximum).

Critical moves to test were the 3 checking potential keymoves:

1. g8=N+? [Refuted after 13 days of analysis with a single Gustav in full-hash mode]

1. h8=R+? [Needed human assistance – Took about 55 days to be completed]

1. h8=Q+! [Needed human assistance – Took about 70 days to be completed]

In any case, **even non-checking moves have been tested** from initial position and in every split, until the very last branch has been completely explored and refuted.

THE INTERESTING PART

The main focus of this verification process was the continuous interaction between human strategic understanding and brute-force raw calculation. Contrary to what one could imagine, a long selfmate is not solved simply by "letting the engine think". In highly saturated positions such as this one, the branching factor can explode dramatically, and even a strong dedicated solver may spend enormous amounts of time exploring strategically irrelevant continuations.

This is not a weakness of Gustav itself, but rather a natural consequence of the position's structure. The engine follows the logic of its search algorithm consistently and rigorously, while the human analyst may immediately perceive deeper positional priorities that may be not directly encoded in the machine evaluation process.

A typical and extremely instructive example is the Black move ..Kg6xRf7. From a purely material perspective, capturing an entire white Rook appears highly attractive and was therefore often explored very early by the engine. However, in this problem the white Rook on f7 acts as a geometrical "seal": as

long as it remains on the board, many White checks and latent defensive possibilities remain unavailable or illegal (for example, in many lines, a highly complex line beginning with g8=Q+, becomes possible because of the disappeared Rf7; otherwise, it would be immediate checkmate). Capturing it too early actually enlarged the tree of legal continuations, producing an explosion of additional branches that were mostly irrelevant to the true thematic content of the problem. For this reason, the analysis often required manual supervision and strategic splitting of the position. The engine had to be guided toward more forcing continuations first, postponing moves like ...Kxf7 as long as possible, until all stronger checking or key-pieces capturing resources had been exhausted. In practice, this means that human intervention was not used to "assist" the engine toward a desired conclusion, but rather to reduce combinatorial noise and keep the brute-force process computationally realistic, even with a "normal" hardware.

Below are listed the specified aspects which I would treat separately, even if they are undoubtely deeply interconnected.

First attempts to simplification

An instructive example appeared during the verification of non-checking moves. Although many of them were strategically trivial, a complete brute-force test still had to consider all of them as fully legal candidates. In many branches, the move ..Rxe2+ (when still possible) became the ideal splitting point: after the almost immediate refutation of Kd1 (on the most cases), where ..Bxc2+ initiates a forced sequence causing the position to collapse within minutes, the analysis could then focus entirely on the only remaining continuation, Kf1, allowing a far more efficient continuation of the search process. For example, instead of ..Kxf7, a frequent choice was ..Rxc2!, preparing a forced checking sequence ..Bd3+ ..Rg2+, or at least permitting ..Rxc5 often followed by ..Rxa5, with significative simplification of the position. Also helped the possibility to find transpositions between forced variations belonging to different universes. A good example in some lines is the sacrifice Qh6+/Rh6+, which is refuted by the obvious Kxh6, so it's enough to calculate just one and save, possibly, a day of useless extra-calculations. Also, other minor transpositions in micro-splits could be reused. To find the better

option sometimes required to use the progressive calculation method, "Zugzahl Unbekannt", which shows precisely the multiplicative factor of time and complexity. Testing various moves even for few minutes could give an important hint about the most promising candidates. I found myself changing the first move proposed by the engine rather often: sometimes worked very well, sometimes not.

Another way to simplify the analysis in the early stage was to keep one Gustav instance (or two, if possible) permanently dedicated to controlling the weak defense .. Rxe2+ Kd1. In some cases, the refutation was not obtained within a few minutes (e.g. when wRa1 controls square c2, partially preventing a free check with ..Bxc2+, or in the brute-forcing from initial position, with a move count still high), so while the other Gustav instances were working on harder schemes with more memory, I could repeatedly test this weak defense against every move in a position, optimizing the overall verification process and saving precious time.

Managing the structural weakness of the castling refutation

Another crucial aspect of this specific problem was

the management of long castling. Due to a well-known characteristic of Gustav's search behavior, a White castling move is generally followed by a comparatively weak Black defense. This defense, for what I experienced in years of Gustav usage, is never a check, can only be a capture if performed by the King itself, and in most cases consists of a neutral move (typically a quiet King or pawn move, sometimes a piece move). With this behaviour, the engine overlooks also variations which would immediately seal the position with a perpetual check. This created a major practical issue: whenever castling was admitted into a variation, a highly valuable tempo was effectively "wasted" on an almost harmless Black move, which at best permitted to run out from some checks. By using splitting strategies that restricted castling possibilities (...Rxe2+ was particularly effective in this regard, making a white King's move mandatory), or by analyzing castling keymoves separately (since castling is always analyzed as the final candidate move), it became possible to eliminate a large number of irrelevant branches while maintaining complete certainty that no hidden variation was being overlooked.

Splitting is annoying - but speeds up a lot

During the verification, several branches that appeared computationally intractable when analyzed as part of a larger search tree became manageable once isolated through manual splitting. This suggests that, for extremely large selfmate trees, dedicating hash resources to a single critical branch may be substantially more effective than sharing the same memory across numerous competing continuations. Gustav does not simply overwrite transposition table entries when memory becomes full. Instead, positions are assigned an internal priority value reflecting their expected usefulness. Once the hashtables reach maximum capacity, only entries considered less valuable than newly generated positions are replaced. This behaviour helps to explain why analysis quality remained relatively stable even after nominal memory saturation, and why manual splitting often proved beneficial by concentrating the available hash resources on a single branch rather than forcing unrelated variations to compete for the same protected entries.

Material reduction – not every man on the board is equivalent

Another unexpected lesson emerged from the progressive destruction of material during the verification process. At first sight, one could assume that the computational complexity of a branch decreases more or less proportionally to the amount of material remaining on the board. Practical experience suggested that the reality is far more subtle. Certain pieces acted as genuine generators of combinatorial complexity.

In particular, a white Bishop on f3 proved to be one of the most influential pieces in the entire position. Removing it often reduced not only the number of legal moves, but also the number of latent defensive resources, allowing at the same time checking possibilities and geometrical interactions in future stages of the search. In many branches, an early capture of this Bishop produced a dramatic acceleration of the brute-force process. By contrast, removing material that was strategically inactive frequently produced little or no measurable benefit. The effective complexity of the position depended less on the quantity of material than on its capacity to generate future legal continuations. Consider

what happens in case Black plays ...Kxh7; four promotions are eliminated, and if follows g8=Q+ Kxg8, other three are eliminated (since Gustav deletes automatically the other promotions if the refutation is the immediate capture of the promoted piece). The same principle, when possible, has ben applied also to en passant, unifying the c3 – c4 moves in one, when the capture was a good option (typically if executed delivering check to White).

This observation gradually led to a practical principle during the analysis: when several refutations appeared equivalent from a purely chess point of view, preference was often given to the continuation eliminating the greatest amount of future mobility or heavy obstacles, rather than the greatest amount of material. A good example is the "trivial" variation 1. Bxf3 Kxh7: after that, the Black defense 2. ... Rxh3 seemed to be more effective than the immediate capture of the bBf3, because after Rxh3 the bBf3 is still en prise, there is no possible skewer that makes the wRh3 disappear, and practically every wBf3's move is countered by a free check in e3 or h1, while dangerous White moves like Ng1 became impossible (see the last paragraphs for more details, as this discovery leads to unexpected

conclusions).

Move ordering

In highly complex selfmate positions, the quality of Black move ordering may become more important than raw search speed. Several branches that appeared computationally difficult were dramatically simplified when forcing defensive moves were manually prioritized, suggesting that significant improvements could potentially be achieved through more sophisticated defence-generation heuristics.

An example will clarify this point. This configuration appeared once during an analysis: wKe1, bRf2 [undefended], wBe2, bPf3, bBe4. White explores the variation 1. Be2xPf3. What is the better reply? Naturally, you will say, it's Rxf3. It recaptures a White piece, and contemporarily saves the Rook from the capture threat from the black King, yet prepares a disturbing free check at next move with ... Re3+. But Gustav insisted to capture with the Bishop, allowing at some time Kxf2, which slowed the analysis of many hours. Thanks to splitting, I was able to "instruct" Gustav to play the Rook recapture, allowing a faster resolution of the branch. However, I

must note that in the final stage of the analysis, when much bigger hashtables were used (12 GB), this behaviour showed a tendency to change in favour of more appealing moves. This remains, therefore, a not completely clear point.

The "80/20" principle of verification

One recurring pattern emerged throughout the entire verification process.

In theory, a difficult branch may contain dozens of legal continuations that all require investigation. In practice, however, computational effort was rarely distributed uniformly across those moves. A surprisingly small number of continuations typically consumed the overwhelming majority of the total verification time. Repeatedly, branches containing thirty, forty or even more legal moves would behave according to a highly asymmetric pattern. Most continuations could be refuted quickly, often within minutes. Yet one or two specific moves would generate enormous search trees and absorb the vast majority of computational resources.

Informally, the phenomenon resembled a form of Pareto distribution: approximately 80% of the verification effort was frequently spent on less than

20% of the available continuations.

The final Queen branch provided a particularly clear example. Most White moves could be eliminated relatively efficiently, while a small number of highly disruptive continuations repeatedly forced extensive investigation. Among them, moves such as Ng1 proved exceptionally expensive because they disrupted Black's forcing mechanisms and prevented the formation of efficient checks networks.

This observation suggests that the practical difficulty of long selfmate verification is often determined not by the total number of legal moves, but by the existence of a few computationally toxic continuations. Once these critical moves are identified and understood, the structure of the search becomes considerably more transparent (as introduced in the "Material reduction" chapter).

Conversely, a branch may appear intimidating because of the sheer number of legal continuations while actually being relatively easy to verify if none of those continuations generates a large search explosion.

Understanding the search: predicting progress beyond move counts

Another interesting observation made during the verification process was that effective management of long brute-force analyses requires understanding not only the position itself, but also the behavioural patterns of the solving engine.

At first sight, monitoring progress appears straightforward. Gustav continuously reports the move currently under investigation, creating the impression that the remaining workload can be estimated simply by counting unexplored moves.

Practical experience demonstrated that this assumption is often misleading. As said in the previous chapter, in many critical branches a single move could require more computation time than dozens of other legal moves combined. Consequently, the numerical progress indicator alone frequently provided a distorted picture of the actual state of the search. As verification advanced, it became increasingly apparent that Gustav follows a relatively consistent internal logic when selecting and exploring candidate moves. While the exact implementation remains unknown, repeated

observation revealed recognizable patterns in the order of investigation.

This knowledge gradually became a practical tool.

By observing which moves had already been analysed and which moves remained unexplored, it was often possible to estimate the true amount of work still ahead more accurately than by relying on move numbers alone. For example, a branch displayed as "move 20 of 50" might already be 90% complete if the major computational bottlenecks had been eliminated. Conversely, a branch displayed as "move 45 of 50" could still require many hours if one of the remaining moves was known to be exceptionally difficult. This may seem trivial, but it is far from obvious. In fact, this understanding proved particularly valuable when planning overnight calculations. Knowing the approximate position of the major bottlenecks often allowed computational resources to be allocated more efficiently. Instead of launching analyses blindly, it became possible to estimate whether a branch would likely finish during the night or continue running well into the following day. A second benefit emerged from increasing familiarity with the positions themselves.

After weeks of examination, many critical structures became recognizable. When Gustav appeared to be stalled on a specific move, it was often possible to identify exactly which defensive resource was causing the delay and how many potentially difficult continuations remained. This transformed progress estimation from a purely numerical exercise into a positional one.

In other words, the most useful indicator was no longer:

"Which move number is Gustav analysing?"

but rather:

"Which position has Gustav reached, and which difficult ideas still remain unexplored?"

As the verification progressed, the ability to answer this question from an human point of view became increasingly accurate. Therefore, human understanding of both the position and the engine's search behaviour gradually became an active component of the verification strategy. The more familiar I became with recurring tactical motifs, defensive resources, and Gustav's exploration patterns, the more accurately future computation times could be predicted and managed.

Why the Queen branch (1. h8=Q+!) survived so long

Another aspect of the "Move order counts" and "Material reduction" paragraphs can be furtherly developed because of its crucial importance for the overall analysis.

The last surviving Queen-promotion forcing branch [1. h8=Q+ Kg6 2. Bxf3 Rxc2!] proved to be one of the most computationally demanding lines. Its exceptional resistance was not caused by the existence of hidden resources for White, but rather by an unfortunate interaction between tactical correctness and search efficiency. Throughout the verification process, one defensive idea repeatedly emerged as particularly effective for Black: the immediate elimination of White's Bishop on f3. In this critical Queen branch, however, the most natural implementation of this plan failed. An immediate ...Bxf3 **allowed a hidden cook** which did not exist in other branches. Although strategically desirable, the move was therefore objectively incorrect and had to be abandoned. As a consequence, Black was forced to adopt a less efficient move order. The alternative defence began with ...Rxc2, postponing the Bishop capture by one move. Only afterwards could ...Bxf3 be played safely, once the tactical cook had

disappeared.

This "forced" delay gave White an additional tempo to reorganise his pieces before the critical Bishop could be eliminated. In many continuations the Bishop escaped from f3 and was eventually captured on a different square, or not captured at all. When the Bishop survived longer and migrated to alternative squares (for example squares g4, g2 or h1, and in particular the **d1** square, where the white Bishop couldn't legally be captured), or could be protected (Ng1 was a huge obstacle too, as already pointed out) this coordination was often lost. The resulting positions contained actually no forcing continuations, fewer immediate checks (and only at the cost of giving away the Rook) and significantly larger branching factors. Instead of exploring narrow tactical corridors, Gustav was forced to investigate broad families of strategically distinct continuations. In practice, the branch became difficult not because White's position was stronger, but because Black's most favourable tree-compressing mechanism could not be activated immediately. The computational burden was therefore shifted from tactical refutation to exhaustive exploration. In such "extreme" analysis, even a single tempo may have consequences

measurable not in moves, but in many hours (or days!) of additional calculations.

The "Paradox of the last Black piece"

A particularly curious phenomenon concerned the relationship between Black material and search difficulty. One might naturally expect positions with fewer Black pieces to be easier to analyze, becasue of the simple principle "*If Black has no men on the board, a selfmate is not possible*". Surprisingly, this was not always the case. In several branches, positions where only pawns + black Bishop survived proved significantly harder to refute than positions where Black was reduced to pawns only. How this apparent paradox can be explained? I'll give a try.

The explanation appears to be combinatorial rather than tactical. A lone Bishop can still generate numerous geometrical possibilities, long-range interactions, captures and defensive resources, while being insufficiently forcing to collapse the tree quickly. The result is easily a large number of viable continuations with relatively little strategic structure. When Black was reduced to pawns only, however, the search often became considerably simpler. Pawn moves are inherently limited and generate less

branches. In many such positions, the defensive tree collapsed surprisingly fast despite the apparent increased White's freedom.

This was one of the most counterintuitive discoveries of the entire project: reducing material did not always simplify the search, but reducing branching potential almost always did. The analysis also revealed an important distinction between reducing material and reducing depth. At first, much effort was devoted to finding continuations that quickly eliminated dangerous pieces or promotion possibilities. This often proved effective, but not always optimal from a computational perspective. In several critical branches, forcing sequences of checks turned out to be even more valuable than immediate material reduction. A typical example occurred after Black's defensive move ...Rxe2+, where White was often forced to respond with Kf1. Human intuition would naturally suggest eliminating the promotion threat first, by capturing on h7.

However, a more efficient strategy sometimes consisted of playing ...Re1+ immediately, sacrificing the Rook. From a chess perspective, this appears paradoxical: Black voluntarily removes one of its most important pieces. From a computational

perspective, however, the idea can be considered. White has only few legal replies, each leading to at least one second forcing check (...f2+ or ...Bd3+). The resulting sequence rapidly shortens the remaining length of the problem and dramatically reduces the number of branches that must be explored.

This observation led to a practical rule of thumb during the verification: whenever possible, forcing checks were often more valuable than preserving material, especially if the checking pieces have little mobility and no forced sequences but the sacrifice. A reduction of only two remaining moves may decrease the effective search effort by orders of magnitude, whereas a significant material reduction may leave the overall branching structure almost unchanged. In this specific selfmate, where the multiplicative time factor is about x10-x15 at every new depth, this aspect was worth a investigation.

However, here an interesting methodological question remains unresolved.

During the verification, forcing sequences of checks often appeared highly beneficial because they rapidly reduced the remaining length of the selfmate and dramatically restricted the opponent's legal options. In several

branches this strategy produced substantial computational savings. However, it is not entirely clear whether such depth reduction is always advantageous. Compressing the search tree at an early stage may simply postpone combinatorial complexity to later positions where a large amount of material remains on the board. In such cases, the branching factor may subsequently re-expand, partially or completely offsetting the benefits gained through the forcing sequence.

This raises a broader question. The most efficient continuation may not necessarily be the one that minimizes the remaining number of moves, but rather the one that minimizes the total amount of future search. In practice, these two objectives in many cases coincide, but they are not theoretically equivalent.

The present investigation was not designed to answer this question conclusively. Nevertheless, repeated observations suggest that the relationship between depth reduction and overall search complexity deserves further study, particularly in long selfmates where branching factors vary dramatically during the course of the solution.

Principle of deferred recapture and multiple threats

During the verification process, an interesting behavioural pattern repeatedly emerged in Gustav's

defensive play.

When White captured a Black piece that could immediately be recaptured, Gustav frequently chose not to execute the recapture at once. Instead, it often preferred another move and postponed the recapture for one or more moves, even when the captured material remained available later.

Although the internal mechanisms of Gustav are unknown to me, a possible explanation can be proposed. An immediate recapture removes a White piece, but it also consumes one defensive resource. Once the recapture has been played, Black may have fewer active threats available on the following move. By contrast, postponing the recapture may allow Black to maintain two independent sources of pressure simultaneously. One threat is the latent recapture itself; the other may be a future check, a promotion threat, an opening of a line, or another forcing continuation. In such situations White is often unable to neutralize both threats with a single move. This can lead to a situation like this:

- If White saves the attacked piece, the forcing continuation becomes available.
- If White prevents the forcing continuation, the

piece may remain vulnerable to recapture and the overall number of legal moves will decrease.

From a search-tree perspective, this creates a form of delayed compression. Instead of spending a move to simplify the position immediately, Black preserves multiple forcing possibilities whose combined effect may reduce the size of the future tree more efficiently. Several of the most difficult branches encountered during the verification appeared to be connected with the opposite phenomenon. In those lines White could sometimes place a single piece on a strategically important square, simultaneously blocking multiple checking lines or defensive mechanisms. The notorious move Ng1 in several major branches represented a typical example: its strength did not lie in material gain or direct tactical threats, but in its ability to neutralize several independent Black resources at once (both Rook checks in e2 and h1). This suggests that, at least in long selfmates of this type, the computational difficulty of a branch is often related not simply to the number of legal moves, but to the number of independent threats that remain available to the defending side. Branches in which Black can

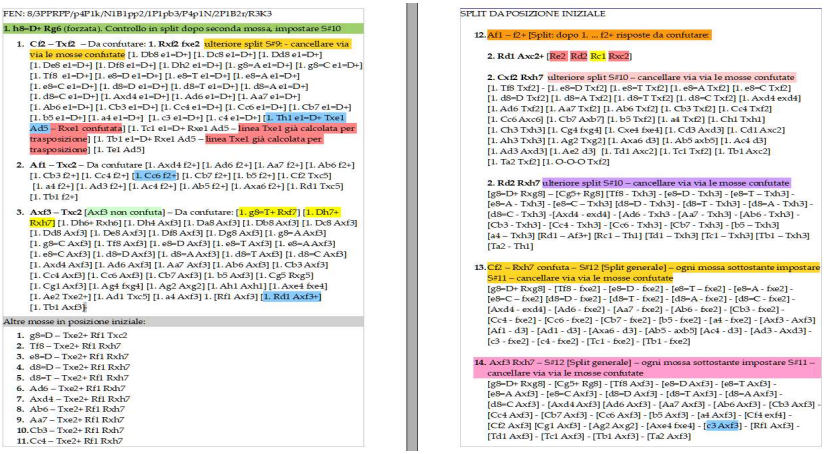
maintain several active threats tend to collapse relatively quickly, whereas branches where White can neutralize many threats with a single move frequently become computational bottlenecks.

While this interpretation remains speculative (I can only suppose that this behaviour could be related to hashtables and their constant interaction with the calculations), it consistently matched the practical behaviour observed throughout the verification.

Verification under adverse conditions

An aspect rarely discussed in theoretical papers, but very relevant in practice, was operational resilience. The analysis was conducted on obsolete hardware, without an uninterruptible power supply during most of the project, in a mountainous area where temporary power failures caused by storms are not uncommon. Consequently, brute-force verification could not be approached as a single uninterrupted computation, at least at early stages. It would have been a suicidal strategy. Then, various forms of redundancy became necessary. Long branches were often attacked simultaneously from different directions. While one Gustav instance explored a position from its root, another could be assigned to a

deeply nested subvariation, with the goal of progressively reducing the unexplored interval between the two analyses. This approach significantly reduced the risk of losing substantial progress after an unexpected interruption. Every progress was immediately taken note whenever possible (unfortunately I had to sleep and to eat, and sometimes even get out of home).



FEN: 8/3PPRPP/p4P1k/N1B1pp2/1P1pb3/P4p1N/2P1B2r/R3K3

1. h8=D+ Rg6 (forzata). Controllo in split dopo seconda mossa, impostare S#10

1. **Cf2 – Txf2** – Da confutare: **1. Rxf2 fxe2** ulteriore split S#9: - cancellare via via le mosse confutate [1. Db8 e1=D+] [1. Dc8 e1=D+] [1. Dd8 e1=D+] [1. De8 e1=D+] [1. Df8 e1=D+] [1. Dh2 e1=D+] [1. g8=A e1=D+] [1. g8=C e1=D+] [1. Tf8 e1=D+] [1. e8=D e1=D+] [1. e8=T e1=D+] [1. e8=A e1=D+] [1. e8=C e1=D+] [1. d8=D e1=D+] [1. d8=T e1=D+] [1. d8=A e1=D+] [1. d8=C e1=D+] [1. Axd4 e1=D+] [1. Ad6 e1=D+] [1. Aa7 e1=D+] [1. Ab6 e1=D+] [1. Cb3 e1=D+] [1. Cc4 e1=D+] [1. Cc6 e1=D+] [1. Cb7 e1=D+] [1. b5 e1=D+] [1. a4 e1=D+] [1. c3 e1=D+] [1. c4 e1=D+] [1. Tb1 e1=D+ Txe1 Ad5 – Rxe1 confutata] [1. Tc1 e1=D+ Rxe1 Ad5 – linea Txe1 già calcolata per trasposizione] [1. Tb1 e1=D+ Rxe1 Ad5 – linea Txe1 già calcolata per trasposizione] [1. Te1 Ad5]
2. **Af1 – Txe2** – Da confutare [1. Axd4 f2+] [1. Ad6 f2+] [1. Aa7 f2+] [1. Ab6 f2+] [1. Cb3 f2+] [1. Cc4 f2+] [1. Cc6 f2+] [1. Cb7 f2+] [1. b5 f2+] [1. Ce2 Txc5] [1. a4 f2+] [1. Ad3 f2+] [1. Ac4 f2+] [1. Ab5 f2+] [1. Axa6 f2+] [1. Rd1 Txc5] [1. Tb1 f2+]
3. **Axf3 – Txe2** [Axf3 non confuta] – Da confutare: [1. g8=T+ Rxf7] [1. Dh7+ Rxh7] [1. Dh6+ Rxh6] [1. Dh4 Axf3] [1. Da8 Axf3] [1. Db8 Axf3] [1. Dc8 Axf3] [1. Dd8 Axf3] [1. De8 Axf3] [1. Df8 Axf3] [1. Dg8 Axf3] [1. g8=A Axf3] [1. g8=C Axf3] [1. Tf8 Axf3] [1. e8=D Axf3] [1. e8=T Axf3] [1. e8=A Axf3] [1. e8=C Axf3] [1. d8=D Axf3] [1. d8=A Axf3] [1. d8=T Axf3] [1. d8=C Axf3] [1. Axd4 Axf3] [1. Ad6 Axf3] [1. Aa7 Axf3] [1. Ab6 Axf3] [1. Cb3 Axf3] [1. Cc4 Axf3] [1. Cc6 Axf3] [1. Cb7 Axf3] [1. b5 Axf3] [1. Cg5 Rxg5] [1. Cg1 Axf3] [1. Ag4 fxg4] [1. Ag2 Axg2] [1. Ah1 Axh1] [1. Axe4 fxe4] [1. Ae2 Txe2+] [1. Ad1 Txc5] [1. a4 Axf3] 1. [Rf1 Axf3] [1. Rd1 Axf3+] [1. Tb1 Axf3]

Altre mosse in posizione iniziale:

1. g8=D – Txe2+ Rf1 Txc2
2. Tf8 – Txe2+ Rf1 Rxh7
3. e8=D – Txe2+ Rf1 Rxh7
4. d8=D – Txe2+ Rf1 Rxh7
5. d8=T – Txe2+ Rf1 Rxh7
6. Ad6 – Txe2+ Rf1 Rxh7
7. Axd4 – Txe2+ Rf1 Rxh7
8. Ab6 – Txe2+ Rf1 Rxh7
9. Aa7 – Txe2+ Rf1 Rxh7
10. Cb3 – Txe2+ Rf1 Rxh7
11. Cc4 – Txe2+ Rf1 Rxh7

SPLIT DA POSIZIONE INIZIALE

12. Af1 – f2+ [Split: dopo 1. ... f2+ risposte da confutare:

2. **Rd1 Axe2+** [Re2 Rd2 Rc1 Rxe2]

2. **Cxf2 Rxh7** ulteriore split S#10 – cancellare via via le mosse confutate [1. Tf8 Txf2] - [1. e8=D Txf2] [1. e8=T Txf2] [1. e8=A Txf2] [1. e8=C Txf2] [1. d8=D Txf2] [1. d8=A Txf2] [1. d8=T Txf2] [1. d8=C Txf2] [1. Axd4 exd4] [1. Ad6 Txf2] [1. Aa7 Txf2] [1. Ab6 Txf2] [1. Cb3 Txf2] [1. Cc4 Txf2] [1. Cc6 Axc6] [1. Cb7 Axb7] [1. b5 Txf2] [1. a4 Txf2] [1. Ch1 Txh1] [1. Ch3 Txh3] [1. Cg4 fxg4] [1. Cxe4 fxe4] [1. Cd3 Axd3] [1. Cd1 Axc2] [1. Ah3 Txh3] [1. Ag2 Txg2] [1. Axa6 d3] [1. Ab5 axb5] [1. Ac4 d3] [1. Ad3 Axd3] [1. Ae2 d3] [1. Td1 Axc2] [1. Tc1 Txf2] [1. Tb1 Axc2] [1. Ta2 Txf2] [1. O-O-O Txf2]

2. **Rd2 Rxh7** ulteriore split S#10 – cancellare via via le mosse confutate [g8=D+ Rxg8] – [Cg5+ Rg8] [Tf8 - Txh3] - [e8=D - Txh3] - [e8=T – Txh3] - [e8=A - Txh3] - [e8=C – Txh3] [d8=D - Txh3] - [d8=T – Txh3] - [d8=A – Txh3] - [d8=C - Txh3] - [Axd4 - exd4] - [Ad6 - Txh3 - [Aa7 - Txh3] - [Ab6 - Txh3] - [Cb3 - Txh3] - [Cc4 - Txh3] - [Cc6 - Txh3] - [Cb7 - Txh3] - [b5 – Txh3] [a4 – Txh3] [Rd1 – Af3+] [Rc1 – Th1] [Td1 – Txh3] [Tc1 – Txh3] [Tb1 – Txh3] [Ta2 - Th1]

13. Cf2 – Rxh7 confuta – S#12 [Split generale] – ogni mossa sottostante impostare S#11 – cancellare via via le mosse confutate [g8=D+ Rxg8] - [Tf8 - fxe2] - [e8=D - fxe2] - [e8=T – fxe2] - [e8=A - fxe2] - [e8=C – fxe2] [d8=D - fxe2] - [d8=T – fxe2] - [d8=A - fxe2] - [d8=C – fxe2] - [Axd4 - exd4] - [Ad6 - fxe2] - [Aa7 - fxe2] - [Ab6 - fxe2] - [Cb3 - fxe2] - [Cc4 - fxe2] - [Cc6 - fxe2] -[Cb7 - fxe2] - [b5 - fxe2] - [a4 - fxe2] - [Axf3 - Axf3] [Af1 - d3] - [Ad1 - d3] - [Axa6 - d3] - [Ab5 - axb5] [Ac4 - d3] - [Ad3 - Axd3] - [c3 - fxe2] - [c4 - fxe2] - [Tc1 - fxe2] - [Tb1 - fxe2]

14. Axf3 Rxh7 – S#12 [Split generale] – ogni mossa sottostante impostare S#11 – cancellare via via le mosse confutate [g8=D+ Rxg8] - [Cg5+ Rg8] - [Tf8 Axf3] - [e8=D Axf3] - [e8=T Axf3] - [e8=A Axf3] - [e8=C Axf3] - [d8=D Axf3] - [d8=T Axf3] - [d8=A Axf3] - [d8=C Axf3] - [Axd4 Axf3] [Ad6 Axf3] - [Aa7 Axf3] - [Ab6 Axf3] - [Cb3 Axf3] - [Cc4 Axf3] - [Cb7 Axf3] - [Cc6 Axf3] - [b5 Axf3] - [a4 Axf3] - [Cf4 ex4] - [Cf2 Axf3] [Cg1 Axf3] - [Ag2 Axg2] - [Axe4 fxe4] - [e3 Axf3] - [Rf1 Axf3] - [Td1 Axf3] - [Tc1 Axf3] - [Tb1 Axf3] - [Ta2 Axf3]

A photogram of the management of what I thought was the final phase of the analysis

[June 7th, 2026]

An UPS helper was finally gathered during the final stages of verification, where several critical branches

required uninterrupted calculations lasting well beyond twenty hours for each one. At that point, the probability of encountering at least one unexpected power failure became far from negligible. This concern proved justified when a thunderstorm caused a temporary blackout while several Gustav instances were actively analysing difficult branches. The interruption lasted only a few minutes, yet without protection it would have erased almost an entire day of computation. The presence of an Uninterruptible Power Supply (UPS) completely changed the situation. Instead of an immediate system shutdown, the computer remained operational, providing sufficient time to react. All active Gustav instances were immediately paused in order to reduce power consumption and preserve battery autonomy while waiting for electrical service to return. This could not save the analysis from a 8-hours blackout, but in this case I was lucky. The outage lasted approximately five minutes. When power was restored, every analysis remained intact and could be resumed without any loss of progress. This experience demonstrated that, in long verification projects, a UPS should not be regarded as a luxury accessory. It is an integral part of the

computational infrastructure.

Three Gustavs are better than four (on a quadruple-threaded CPU): an unexpected resource contention effect

An empyrical observation made during the verification concerned the relationship between the number of simultaneous Gustav instances and the overall throughput of the system. At first sight, running four parallel instances appeared to be the optimal strategy, as both available computers exposed four logical processors to the operating system. In practice, however, repeated experiments consistently showed that three simultaneous Gustav instances outperformed four, sometimes by a significant margin. The explanation is likely related to resource contention rather than raw computational power. Although the processors remained fully utilized in both configurations, the fourth Gustav instance introduced additional competition for shared hardware resources, particularly CPU caches, memory bandwidth and scheduler time slices.

Gustav's search process, as known, performs a very large number of transposition-table lookups and

repeated accesses to previously explored positions. While the transposition tables themselves reside in main memory, frequently accessed entries are temporarily cached by the processor's L1, L2 and L3 cache hierarchy. When too many independent search processes are executed simultaneously, cache contents are continuously evicted and replaced by data belonging to competing instances. This phenomenon, commonly known as *cache thrashing*, increases the frequency of costly memory accesses and reduces effective search efficiency (cache access time is notoriously orders of magnitude faster than direct RAM access).

Additional overhead is introduced by the operating system scheduler. With four Gustav instances occupying all available logical processors, even minor background activities performed by Windows must compete for execution time. Context switches become more frequent, cache locality deteriorates and each Gustav process receives a slightly less stable execution environment. Although the resulting CPU utilisation remains close to 100%, the amount of useful search work completed per unit of time may actually decrease.

In contrast, running three simultaneous instances

leaves a modest reserve of computational resources available to the operating system. This appears to reduce scheduler interference, improve cache utilisation and increase the amount of effective work performed by each Gustav instance. Empirically, the reduction from four to three parallel searches produced a measurable acceleration of the overall verification process on both tested systems.

Distributed analysis and resource allocation

As the verification progressed, a further methodological evolution became necessary. The initial stages of the analysis could often be handled by a single machine running several independent Gustav instances. However, once the simpler branches had been eliminated, the remaining variations became increasingly computationally demanding. In many cases, a single move could require several hours of analysis, while particularly complex branches could occupy an entire machine for days. At this stage, maximizing raw processor utilization became less important than maximizing the efficiency of available memory and hashtables. Rather than saturating the hardware with the largest possible number of concurrent processes, it often appeared more effective to run fewer Gustav

instances while allocating substantially larger hash resources to each one.

Additional computing resources were gradually integrated whenever available. A second faster (but still relatively modest) computer was employed remotely, through RustDesk, allowing independent branches to be assigned to separate machines. This transformed the verification process into a form of manually managed distributed computation. Critical variations could be explored simultaneously on different systems, while the human operator remained responsible for scheduling, prioritization and branch allocation. **I cannot deny that all this was unexpectedly enjoyable**, like climbing a mountain that does not want to be climbed at all.

An important consequence of this approach was that computational progress became increasingly resilient. The completion of any individual branch contributed permanently to the overall verification effort, regardless of interruptions affecting other machines. As the project expanded, the limiting factor ceased to be processing power alone and became the efficient allocation of available resources across an ever smaller number of exceptionally demanding positions.

In practice, the final stages of the verification resembled less a traditional chess analysis and more a large-scale resource management problem, where understanding which branch deserved computational attention (and how to overcome the issues) became almost as important as the calculations themselves.

Extreme hardware optimization: turning an old i3 into a verification workstation (or at least something like that)

The verification described in this book, as already pointed in the introductory chapter, was not performed on modern high-end hardware.

The primary machine was an Intel i3-4150 3.50 Ghz processor released in 2014, equipped with 16 GB of DDR3 memory at 1600 Mhz frequency and a rather prehistoric HDD storage device. By contemporary standards, this configuration is surely not ideal for intensive calculations, if not clearly obsolete. Consequently, obtaining acceptable performance required extensive optimization.

Since processor overclocking was not available (it was auto-implemented only on the helper

computer), efforts focused on maximizing the efficiency of the existing hardware. The operating system was progressively stripped of every non-essential component that could consume processor time, memory, storage bandwidth, or background resources. Services unrelated to the verification process were either disabled or removed after careful evaluation of their strict necessity. Particular attention was paid to automatic update mechanisms, background maintenance tasks, telemetry-related services, and other processes capable of generating unpredictable activity during long calculations. "Ultimate Performance" settings of Windows 10 were discovered, enabled with the help of a Command Prompt script, and put on use. Windows Defender was temporarily disabled during the most demanding phases of computation in order to reduce memory consumption and eliminate background scanning activity. Additional optimizations included the activation of *large memory pages*, reduction of background applications, elimination of unnecessary startup processes, and extensive cleanup of the operating system environment.

Security mitigations introduced for certain

speculative execution vulnerabilities were also disabled after evaluating the performance implications on this dedicated verification machine. Although such mitigations remain important in normal security-sensitive environments, they introduced measurable computational overhead on older processor architectures.

The result was not a faster processor in the literal sense. Clock frequency remained unchanged, as the number of instructions processed for each cycle. What changed was the proportion of processor time devoted to useful work. Over time, the system evolved from a general-purpose desktop computer into a highly specialized verification workstation whose sole purpose was the elimination of difficult selfmate branches. A fairly atypical use. An interesting detail: core temperatures, even under the highest effort, never got out of control, staying around 70 °C.

The experience produced an interesting output: when modern hardware is unavailable, significant performance gains can often be achieved not by increasing computational power, but by systematically removing every obstacle that prevents existing computational power from being fully

utilized. In practice, the old i3 did not really become a modern workstation. It simply became a machine almost entirely dedicated to one task. It was estimated, consulting various sources, that the mixture of various optimizations speeded up the whole project by a very satisfying value of **+15% / +20%**, without any financial impact (no money was spent in all this process, except for electricity!).

Managing CPU and processes

Two of the most important features of the optimization were, undoubtely, process prioritizing and thread management. As the verification progressed, indeed, a new bottleneck gradually emerged. During the early stages of analysis, the primary challenge was simply the enormous size of the search tree. Later, however, after most easy branches had been eliminated, only a handful of extremely resistant lines remained. At that point, the problem was no longer how to distribute computing power evenly, but how to concentrate it where it mattered most. A crucial information was that Gustav is a strictly single-threaded program. Regardless of the amount of memory assigned to it, each instance can utilize only one logical processor. Consequently, running multiple simultaneous

instances does not create parallel cooperation within a branch, but rather independent searches competing for hardware resources.

The main workstation had 2 physical cores. Although Windows System Information reported four logical processors, a closer examination revealed the actual topology, showing a HyperThreading technology:

Physical Core	Logical Processors
Core 0	CPU 0, CPU 1
Core 1	CPU 2, CPU 3

This distinction proved to be worth of analysis.

The final verification phase contained one extremely critical branch that absorbed the overwhelming majority of the remaining computational effort. This branch was assigned a dedicated Gustav instance with 12 GB of hash memory and High process priority. More importantly, it was permanently bound to CPU 3 through processor affinity settings.

Since CPU 2 and CPU 3 belong to the same physical core, and CPU 2 remained intentionally unused by

Gustav, the critical process effectively received an almost exclusive physical core. The remaining logical processor (CPU 2) was reserved for Windows, background services, RustDesk remote management, web browsers and any unavoidable operating-system activity. Meanwhile, when the need of resources was not so compelling, two secondary Gustav instances were assigned to CPU 0 and CPU 1. These two logical processors belonged to the same physical core and therefore competed for execution resources, cache bandwidth and scheduling opportunities. However, this was considered acceptable because they were analysing relatively minor branches whose completion time was not strategically important.

The guiding principle was simple: if contention must exist somewhere, it should affect non-critical analyses rather than the branch capable of delaying the entire project. This arrangement effectively transformed the processor into two distinct computational zones:

- A dedicated verification core for the most difficult branch.
- A shared work core for secondary analyses and auxiliary tasks.

The approach resembles resource prioritization techniques, traditionally used in larger computing environments, where bottleneck jobs receive preferential treatment while less important workloads share remaining resources.

An additional benefit emerged from this configuration. By reserving one logical processor for operating-system activity, Windows no longer interrupted the most demanding analysis whenever background tasks awakened. Instead, those interruptions were largely absorbed by the otherwise idle sibling thread, reducing interference with the critical search.

The practical result was a small but noticeable increase in overall throughput during the final stages of verification. Rather than maximizing the number of simultaneous Gustav instances, the objective became maximizing the progress of the branch most likely to determine the completion date of the entire project.

Abandoning inefficient searches

One of the most counterintuitive lessons of the verification process concerned the deliberate interruption of ongoing analyses. At first sight,

terminating a Gustav search after several hours of computation may appear wasteful. However, practical experience repeatedly showed that continuing an inefficient branch was often more expensive than abandoning it and restarting from a strategically superior defensive choice.

A recurring example involved positions in which Black had been reduced to a single Bishop and a few pawns. In such situations, as pointed earlier, the defensive resources available to Black became severely limited. The Bishop frequently spent many moves searching for useful targets, without generating forcing continuations, checks, or multiple simultaneous threats. From the perspective of brute-force verification, these positions were often computationally toxic. In several cases, an alternative defensive move discovered later would preserve additional material, maintain more forcing possibilities, or create stronger long-term pressure. Although switching to the new defence required abandoning hours of previous analysis, the resulting reduction in search complexity frequently compensated for the lost computation time and ultimately accelerated the overall verification. This led to an important methodological conclusion: the

objective is not to preserve previously invested computational effort, but to **eliminate the branch as efficiently as possible.**

Consequently, interrupting an ongoing search should not be automatically regarded as a failure. If a newly discovered defensive resource is likely to produce a significantly smaller search tree, abandoning the previous analysis may represent the most rational use of available computational resources. In long selfmates, where a single branch can require many hours or even days of verification, the ability to recognize and discard inefficient searches becomes an important practical skill. Computational persistence remains valuable, but persistence directed toward the wrong branch can easily become counterproductive.

When immediate compression becomes a trap – the ultimate change of approach that allowed to actually complete the project

Maybe the most unexpected lesson of the entire verification project emerged only during its final phase, and it's strictly connected to what explained in the previous section.

Throughout most of the analysis, the guiding principle seemed obvious: whenever several defensive possibilities existed, priority should be given to the move that immediately reduced White's freedom and contemporarily increased Black's one as much as possible. This was also broadly consistent with Gustav's own behaviour. The program, as I could observe, often preferred forcing moves that produced an instant reduction of the search tree, even when alternative continuations remained available. However, the deepest surviving branches eventually revealed a limitation of this approach.

In several critical lines, Gustav repeatedly selected the move ...Rh1+, sacrificing the Rook in exchange for an immediate forcing effect. From a local perspective, the decision appeared perfectly logical. White's replies became highly restricted, and the tree

temporarily contracted.

Yet something remarkable happened afterwards.

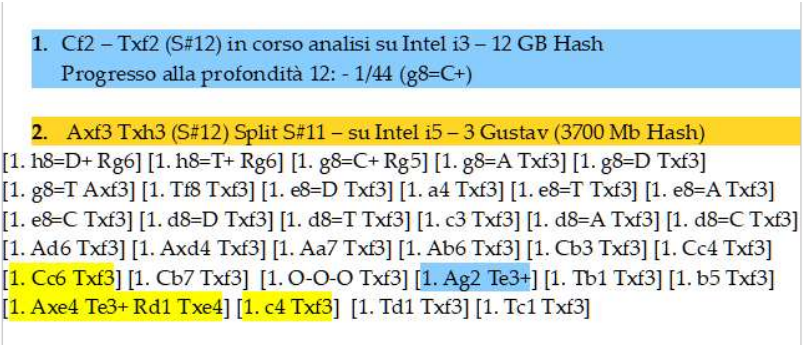
Once the Rook disappeared, Black was often left with little more than a Bishop trapped in the corner and a handful of scattered pawns. The short-term gain was undeniable, but the long-term consequences proved disastrous. Black's attacking resources were again severely reduced, coordination deteriorated, and the search tree expanded again, often by an order of magnitude greater than the initial reduction. So, what's the point? My observation lead to an empyrical conclusion: **the compression was real, but only temporary**. In practice, the move solved an immediate problem while creating a much larger computational problem a few moves later.

These observations led to a radical change of strategy.

Instead of searching for the defence that produced the strongest local forcing effect, attention shifted towards continuations capable of preserving Black's attacking potential for as long as possible. The goal was no longer immediate compression, but "sustainable compression", as I named it. A typical

example arose after the move 1. Bxf3 Rxh3. At first sight, the continuation seemed less forcing than the direct Rook sacrifice. Yet it possessed an important strategic advantage: Black retained both major attacking pieces and preserved several independent threats simultaneously. The Bishop remained active. The Rook remained active and immune to skewers on H-file. The possibility of ...Rxf3 remained available (and in the splits it was played whenever possible, except for the presence of an immediate forcing check not sacrificing material). The free check resource ...Re3+ could often be prepared after ...Rxf3 (an annoying problem, however, was the White move Nc4). At this time, Gustav did not miss that powerful blow in the majority of the lines. In other words, rather than consuming attacking resources immediately, Black continued accumulating pressure, as a principle I already explored in a previous chapter.

This distinction appeared functional. The objective was not merely to reduce the branching factor at the current node, but to avoid situations in which the tree would re-expand dramatically a few moves later. As a result, the final phase of the project developed along two completely different paths.



Fighting the last two bosses (June 24th, 2026)

When just two lines survived, on one computer, a difficult but still manageable branch was allowed to run continuously with maximum resources, following a traditional full-depth verification approach. It was strictly a single-Gustav session, in order to minimize cache misses. On the second machine, an almost completed line (1. Bxf3 Kxh7?!, do you remember?), **only a few moves away from verification, was deliberately abandoned**. Despite the enormous amount of previous work invested in it, persistent computational toxicity suggested that a completely different defensive scheme might ultimately be faster. The apparently good intuition that ... Rxh7 would have prevented many promotions, was probably wrong on a wider

perspective, and it was a sad discovery, when I was apparently so close to complete the task. Actually, permitting to the King to reach 7th rank permitted variations with a major promotion in d8 and e8, followed by another major promotion (or better, four ones, since wRf7 leads to discovered check!) in g8, that consequently could not be reduced to a single one. This caused unconstrained explosion of the analysis tree. And ironically, the most toxic move at all was the already cited Ng1! In "normal" conditions, it was an accident for one move only. But when played immediately, it became something like 50 accidents all together. All the Black coordinations were ruined thanks to one really stupid move. That was another excellent reason why, in the line 1. Bxf3 Kxh7, with only 5 candidates left on more than 40 (d8/e8=Q/R, Ng1), the radical abandonment was actuated. Remember the paragraph "Abandoning inefficient searches" ? This was an extreme example, and I believe it's not casual that happened exactly at the end, when only intractable variations survived. That's why I switched to 1. Bxf3 Rxh3!?, which had a curious characteristics: no toxic moves were found in first instance, because every possibility was already encountered and refuted before. The only three

checking keymoves were the same already certified: 1. g8=N+, 1. g8=R+ and 1. g8=Q+. This permitted to walk on a widely known path and terminate the analysis in efficient time.

At least, it was my thought. And personal thoughts have a bad habit: they are often convincing, logical and coherent ... and still wrong.

Human-Created Transpositions: Designing Future Convergence

One of the most unusual optimisation techniques developed during the final phase of the verification project did not involve hardware, software configuration, hashtables, processor affinity, memory management, or any other technical improvement. Instead, it emerged directly from understanding the structure of the surviving analysis tree itself.

Traditionally, transpositions are considered a passive phenomenon. A solving engine explores different move orders and eventually discovers that two apparently unrelated branches lead to the same position. When this happens, previously calculated information can be reused and substantial amounts of computation are saved. However, during the final stages of this project, a different concept gradually

emerged: transpositions could sometimes be created deliberately rather than merely discovered.

The idea originated while analysing several of the most difficult promotion branches in the newborn 1. Bxf3 Rxh3 line, as shown in the diagram above "*Fighting the last bosses*". Split or not the worst variations in that line? Initially, the natural tendency was to search for the objectively strongest defensive continuation after each White move. This approach appears self-evident. If one defence is stronger than another, it should theoretically reduce White's options more efficiently and therefore produce a faster verification. Yet practical experience repeatedly demonstrated that local optimization and global optimization were not necessarily the same thing.

A move that appeared strongest within a particular branch could simultaneously generate entirely new families of positions elsewhere in the analysis tree. Conversely, a move that seemed slightly less efficient locally could sometimes force future branches to converge into positions that had already been analysed.

A striking example occurred in the promotion line

beginning with:

2. h8=Q+ Kg6

Then, after the majority of White legal moves, several defensive continuations were available to Black. From a purely local perspective, immediate captures on other targets often appeared attractive, like the already cited ..Rxf3. Nevertheless, by systematically choosing continuations ending with ...Rxh8 whenever this remained feasible, an unexpected effect appeared.

Entire sections of the later promotion line:

2. h8=R+ Kg6

began collapsing into already verified positions.

The reason was simple. After moves such as:

Qh7+ Kxh7- Rh7+ Kxh7

the sacrificed piece disappears completely from the board. The resulting position is identical. The same phenomenon occurred after:

Qh6+ Kxh6 / Rh6+ Kxh6,

or Qa8 Bxa8 / Ra8 Bxa8

as was already cited in one of the earliest paragraphs, like some "gifted" occasional benefits.

But this time there was a difference: this "occasional" help could be turned into something sistematic.

Actually, **if every time that the defense ... Rxh8 is possible and is forced through a human-guided split, a new transposition has been formed on the 1. h8=R+ line.** In many individual positions, ...Rxh8 is probably not the strongest local defence available. But if this principle is applied constantly, it leaves the Rook line with only **nine** survived sub-variations to verify, on a total of 53 legal moves.

This means that from the perspective of future verification, the distinction between Queen promotion and Rook promotion nearly vanished. Different branches became identical computational problems. Wonderful discovery, you may say!

A visual diagram will make all this more comprehensible, showing the predicted output after completing the split. It was extremely encouraging to have "found" a way to bypass an obstacle that seemed immune to every attempt to remove it, and the desperation of being stuck quickly turned into new enthusiasm. Those dozens of red-marked variations really impressed me.

[1. h8=D+ Rg6]
Split S#10 su Intel i5 (Hash 3800 Mb con 3 Gustav – 5500 Mb Hash con 2 Gustav)
[1. g8=T+ Rxf7] [1. Dh7+ Rxh7] [1. Dh6+ Rxh6] [1. Dh5+ Txh5] [Ah5+ Txh5]
[1. Da8 Axa8] [1. Db8 Axf3] [1. Dc8 Axf3] [1. Dd8 Axf3] [1. De8 Axf3] [1. Df8 Axf3]
[1. Dg8 Axf3] [1. Dh4 Txh4] [1. Dxh3 Axf3] [1. g8=A Txh8] [1. g8=C Txh8]
[1. Tf8 Txh8] [1. e8=D Txh8] [1. e8=T Txh8] [1. e8=A Txh8 [1. e8=C Txh8]
[1. d8=D Txh8] [1. d8=T Txh8] [1. d8=A Txh8] [1. d8=C Txh8] [1. Axd4 Txh8]
[1. Ad6 Txh8] [1. Aa7 Txh8] [1. Ab6 Txh8] [1. Cb3 Txh8] [1. Cc4 Txh8] [1. Cc6 Txh8]
[1. Cb7 Txh8] [1. b5 Txh8] [1. Ag4 Txh8] [1. Ag2 Txh8] [1. Ah1 Txh8] [1. Ae2 Txh8]
[1. Ad1 Txh8] [1. Axe4 Txh8] [1. a4 Txh8] [1. c3 Txh8] [1. c4 Txh8] [1. Rd2 Txh8]
[1. Rd1 Txh8] [1. Re2 Txh8] [1. Rf2 Txh8] [1. Rf1 Txh8] [1. Td1 Txh8] [1. Tc1 Txh8]
[1. Tb1 Txh8] [1. Ta2 Txh8] [1. O-O-O Txh8]

[1. h8=T+ Rg6]
Split S#10 su Intel i5 (Hash 3800 Mb con 3 Gustav – 5500 Mb Hash con 2 Gustav)

[1. g8=T+ Rxf7] [1. Ah5+ Txh5] [1. Th7 Rxh7 trasposizione]
[1. Th6+ Rxh6 trasposizione] [1. Th5 Txh5 trasposizione]
[1. Ta8 Axa8 trasposizione] [1. Th4 Txh4 trasposizione]
[1. g8=A Txh8 trasposizione] [1. Txh3 Axf3]
[1. g8=C Txh8 trasposizione] [1. Tff8 Txh8 trasposizione] [1. Tb8 Axf3]
[1. e8=D Txh8 trasposizione] [1. e8=T Txh8 trasposizione] [1. Tg8 Axf3]
[1. e8=A Txh8 trasposizione] [1. e8=C Txh8 trasposizione] [1. Thf8 Axf3]
[1. d8=D Txh8 trasposizione] [1. d8=T Txh8 trasposizione] [1. Te8 Axf3]
[1. d8=A Txh8 trasposizione] [1. d8=C Txh8 trasposizione] [1. Td8 Axf3]
[1. Axd4 Txh8 trasposizione] [1. Ad6 Txh8 trasposizione] [1. Tc8 Axf3]
[1. Aa7 Txh8 trasposizione] [1. Ab6 Txh8 trasposizione] [1. Cb3 Txh8 trasposizione]
[1. Cc4 Txh8 trasposizione] [1. Cc6 Txh8 trasposizione] [1. Cb7 Txh8 trasposizione]
[1. b5 Txh8 trasposizione] [1. Ag4 Txh8 trasposizione] [1. Ag2 Txh8 trasposizione]
[1. Ah1 Txh8 trasposizione] [1. Ae2 Txh8 trasposizione] [1. Ad1 Txh8 trasposizione]
[1. Axe4 Txh8 trasposizione] [1. a4 Txh8 trasposizione] [1. c3 Txh8 trasposizione]
[1. c4 Txh8 trasposizione] [1. Rd2 Txh8 trasposizione] [1. Rd1 Txh8 trasposizione]
[1. Rf2 Txh8 trasposizione] [1. Rf1 Txh8 trasposizione] [1. Re2 Txh8 trasposizione]
[1. Td1 Txh8 trasposizione] [1. Tc1 Txh8 trasposizione]
[1. Tb1 Txh8 trasposizione] [1. Ta2 Txh8 rasposizione]
[1. O-O-O Txh8 trasposizione]

Variations colored in red are NOT to calculate anymore!
(June 25th, 2026)

Even if the Queen branch became with all probability more expensive to verify, the expected saving obtained by reducing the Rook branch from 53 legal moves to only nine surviving independent variations appeared overwhelmingly favourable. The objective, then, was no longer merely to verify a single branch as efficiently as possible. Instead, it became possible to deliberately shape the structure of future analysis by selecting continuations that maximised convergence between different branches. In effect, a form of "**human hashtable**" was created. Gustav could only recognise transpositions after reaching them. The human observer, having accumulated extensive familiarity with the position, could sometimes predict those transpositions several moves in advance and guide the analysis towards them, even before the calculations even started.

Of course, my so-called "human-created transpositions" were thought to be applied selectively rather than systematically. If a branch could be refuted within a few minutes, forcing convergence often produced less benefit than simply allowing Gustav to solve both positions independently. Like every optimisation technique, manual transpositions carried an administrative cost,

and therefore are worthwhile only when the expected computational savings clearly exceeded the additional human effort required to manage them.

This approach also represented a natural extension of another principle developed during the project: "sustainable compression". Earlier chapters discussed the difference between immediate reduction of the search tree and long-term reduction of the search tree. Human-created transpositions followed exactly the same philosophy. The goal was not necessarily to minimise the current branch. The goal was to minimise the total amount of verification work that still remains.

However, this approach was not so simple and efficient as it seemed. When you are making a homemade research, starting from a generic idea and without a well-defined method, but instead you learn step by step, surprises come out relentlessly. And here comes the definitive plot twist. "Promising" is different from "Real". In other words, **forcing ..Rxh8 for almost every move in the Queen variation proved soon to need an incredible amount of time of analysis**. Factors were so high to make me doubt of my own idea, even if I remained convinced that the principle was good. So, what to

do? Continue on the last 5 awfully toxic branches of the 1. Bxf3 Kxh7 variation, hoping that a hyper-boosted processor and big hashtables could climb the Olympus, or try the new road, hoping that my new experimental strategy would help somehow? What should you do if the cure seems worst than the disease itself? Well, I needed some practical data in order to decide.

When I radically changed the approach, for the first days I was actually observing strange behaviours from Gustav. Some lines that normally collapsed in few hours in every possible variation, became some sort of intractable bottlenecks. Maybe the defense ... Re3+ was not so good or forcing as I thought. Expecially the trivial variation **1. b5?** proved to be exceptionally resistant to verification. But why? That move never created problems. It always was one of the moves I was able to discard quickly. This was a first signal. Then, I checked out with raw numbers what my change of strategy would have really meant.

Partial tests soon revealed an uncomfortable reality. The idea was working exactly as intended. Transpositions were indeed being created and dozens of branches eliminated. Unfortunately,

creating them proved extraordinarily expensive. Several key branches, necessary for this approach, displayed multiplicative growth factors comparable to, and sometimes even exceeding, those observed in the infamous Ng1 variation. Panic! For a moment, a thought crossed my mind: "*This analysis will never end. It's like cutting off one of the Hydra's heads. In a few seconds, two new heads immediately grow back".*

So, all considered, I took a step back. **Again, I changed my mind, for the same principle that made me change my mind just a few days before: abandoning inefficient procedures.** This is part of the research progress, especially when you are not a professional researcher. The decision was not taken because of the amount of analysis already done: it was taken only because what seemed to be a wonderful idea was rapidly turning into a worse nightmare.

The verification, thus, ended in this way:

- The Intel i3 was dedicated exclusively to defeat the 1. Nf2 Rxf2 variation, in a colossal assault with full hash, while attacking the minor branches from behind, trying to make the two analysis meet in the middle.

- The Intel i5 fought with those 5 toxic moves one by one, relying on the turbo-boost and the more modern architecture, also in full hash mode. It was only a matter of time, patience and electrical power.

1. Cf2 – Txf2 (S#12) analisi su Intel i3

S#11 [1. h8=D+ Rg6] [1. h8=T+ Rg6] [1. g8=D Txe2+] [1. g8=T Txe2+]
[1. Tf8 Txe2+] [1. e8=D Txe2+] [1. d8=D Txe2+] [1. Ad1 Te2+] [1. Af1 Txf1+]
[1. Axf3 Txf3] [1. Axa6 Te2+] [1. Ab5 Te2+] [1. Ac4 Te2+] [1. Ad3 Te2+]
[1. Rxf2 fxe2]

2. Axf3 Rxh7 (S#12) analisi su Intel i5

S#11 [1. Cg1 Axf3] [1. e8=D Axf3] [1. e8=T Axf3] [1.d8=D Axf3]

Cg1 Axf3 - SPLIT S#10
[1. Tf8 Th1] [1. e8=D Th1] [1. e8=T Th1] [1. d8=D Th1] [1. d8=T Th1]
[1. Axd4 exd4] [1. Cb3 Th1] [1. b5 Th1]

The ultimate effort to climb the highest peak

[June 26th, 2026)

Ironically, what seemed to be the most dangerous monster, Ng1, **in the end proved to be the easiest variation to refute**: The entire difficulty ultimately depended on a single positional detail that I had initially overlooked. Actually, it was enough to change, through a last aimed split, the bad defense ..Re2+ (always the Gustav's favourite) with

the strong pin ..Rh1! The search tree, under these circumstances, collapsed in a couple of days without any substantial difficulty. The lines in which a heavy piece (Queen or Rook) comes on 8th rank were more complex, because of the various series of checks they allowed, but the structural weakness of the pinned Ng1 was something permanent, ready to surface as soon as the forcing sequence came to an end.

Even more ironically, the human hashtables were actually used, but in the other analysis line. After 1. Nf2 Rxf2, many legal Bishop moves are strategically equivalent: 2. Bd1, Bf1, Bb5, Bc4, Bd3. That because, if played 2. ... Re2+ 3. Bxe2, the resultant position is identical. All that I had to do was exclude all the secondary variations with the refusal of capture in e2 (and it was quick, especially for 2. Bd1 and 2. Bf1 which reduced the flight squares to only one), and then unify all those variations in one, calculating the Bxe2 configuration only one time. Obviously, this was not applicable to moves like 2. Bxa6 or 2. Bxf3, since they capture a pawn and then they change, even if slightly, the architecture of the position.

This unexpected outcome also provided the strongest practical confirmation of the ideas developed during the research.

THE FINAL BLOW

June 30th, 2026. The last and apparently strongest boss of the entire project eventually disappeared not through an additional 30-days brute-force computation, but through a ***structural transposition argument.***

The point was that the calculation of 1. Nf2 Rxf2 2. Kxf2 fxe2 line was ***not necessary****. Once the surviving moves were partitioned according to whether the Bishop on e2 moved or not, every remaining branch was shown to be either already refuted directly or equivalent to a position that had already been certified.*

In other words, once it was shown that Kxf2 followed by ...fxe2 always led to a position already contained in previously certified branches (with the help of retrospective analysis of the splits), the remaining unexplored subtree disappeared entirely. After months of calculation, optimisation and human guidance, the last barrier was removed by recognising that the problem had already been solved in a different move order.

Here follows the demonstration process.

THEOREM

The branch beginning with 1. Nf2 Rxf2 2.Kxf2

does not generate any new search tree and therefore requires no independent brute-force verification.

LEMMA

During the exhaustive verification, from starting position of the problem, of the position after

1. Nf2 Rxf2,

every legal White first move except 1. Kxf2 was explicitly analysed.

Whenever White delayed the capture on f2 while keeping the Bishop on e2, the continuation

...fxe2

remained available and produced exactly the same position that arises after

Kxf2 ...fxe2

followed by the same White move.

Conversely, whenever White chose a move allowing Black a stronger tactical resource (such as ..Rxe2+, ...Re2+, or equivalent continuations), those branches were already completely verified and refuted directly.

PROOF

Every legal White continuation after

1. Nf2 Rxf2

belongs to one of the following two mutually exclusive classes.

Case 1.

White keeps the Bishop on e2.

Then the move Kxf2 can still be played before any structural change occurs.

Black's natural reply is ...fxe2, after which the resulting position is identical to one already reached during the exhaustive verification of another split.

Therefore no new subtree is created.

Case 2.

White moves the Bishop away from e2 or otherwise allows Black a stronger tactical continuation.

These branches were already calculated explicitly.

In those cases Black never needed the transposition because stronger defences were already sufficient to refute the line.

Since these two cases exhaust all legal White continuations, no unexplored search tree remains after

1. Nf2 Rxf2 2.Kxf2.

Hence an independent verification of this branch would duplicate positions already certified elsewhere.

Curiously, the final mathematical simplification of the project did not originate from a planned investigation. It emerged almost accidentally while two independent analyses were running simultaneously on two different computers.

On one screen, Gustav was analysing the split beginning with

1.h8=R+ Kg6

where the sequence

2. Kxf2 fxe2

naturally appeared as a branch to be calculated.

On the other screen, another split had just reached

1.Kxf2 fxe2

after which the promotion

2. h8=R+ Kg6

became the critical continuation, where the engine was stuck in a bottleneck.

The two positions looked unexpectedly familiar. The only apparent difference was the order in which the moves had been played.

At that moment it became evident that they were not merely similar positions, but the very same position reached through two different move orders. The supposedly unexplored branch was already embedded inside another search tree that had been analysed independently.

Ironically, the relatively modest speed of the available hardware probably made this observation possible. Had the faster machine completed its task much earlier, the two analyses would never have been visible simultaneously. The coincidence did not prove the theorem by itself, but it revealed the structural relationship that ultimately led to the formal proof.

This was perhaps the clearest example of what may be called **computational serendipity**: a discovery

that arose not from searching directly for a theorem, but from recognising an unexpected pattern while observing independent computations evolving in parallel.

In the end, the last "monstrous" branch of the project disappeared not because it had finally been calculated, but because it had finally been ***recognised****.*

I could hardly have imagined a more fitting ending to this story.

The most important question: why all this?

A natural objection could arise after this journey into computational wonders. In such an extremely forcing selfmate, with Black forces loose and ready to come into play: *why spend enormous computational and human effort, even borrowing a second machine, for the purpose of analysing even the non-checking or absolutely trivial moves, when Black is free to defend in almost arbitrary ways?*

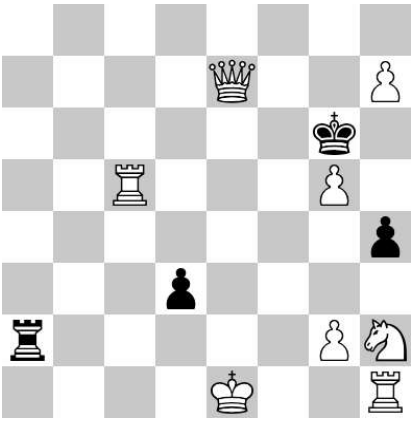
The answer is simple: because "apparently irrelevant" defensive resources are exactly where hidden cooks and duals tend to survive. In long selfmates with highly saturated positions, the intuitive human perception of "forcing play" can become dangerously misleading. A move that appears strategically absurd may nevertheless create a delayed resource, alter the move-order geometry, preserve a critical square, or simply waste a tempo in a way that destabilizes the intended mechanism many moves later. Some of the most dangerous refutations in complex problems are born precisely from moves that a human analyst would instinctively discard without calculation. Probably the best example is the large capture Qh8(Rh8)xRh2, which in many lines still leaves Black with numerous

legal options, but with very limited capacity to counter White's plans.

From a computational perspective, this becomes a true nightmare: the engine must still explore an enormous defensive tree, despite the practical irrelevance of most branches. Paradoxically, this same capture also appears in the actual solution itself: 10. Qxh2!! is one of the deepest and most counterintuitive moments of the problem. White voluntarily removes Black's last active piece, apparently granting maximum defensive freedom, yet this is precisely what creates the final zugzwang mechanism, that cannot be countered in any way. Many times the engine explored similar variations and had to change the Black defense after having encountered a cook, because of this recursiveness.

If this explanation does not convince the reader, I can cite a proof that sometimes, in long selfmates, even a quiet move that leaves enormous freedom to Black could be a cook.

Consider the following problem (curiously, the theme was still a Valladao Task, but without Allumwandlung featured).



Selfmate in 11

(**C+** - Tested with Gustav with **Brute Force**)

Composed by: **Daniele Guglielmo Gatti**

Zalokotsky MT 2022 – 6° Honorable Mention

FEN: 8/4Q2P/6k1/2R3P1/7p/3p4/r5PN/4K2R

Thematic Try: **1. O-O?** [zugzwang]

1. ... Re2 ! [the strongest defense] 2. Rc6+ Kh5 3. Qxe2+ dxe2 4. h8=Q+ Kxg5 5. Qf6+ Kh5 6. Kh1! exf1=Q+ 7. Nxf1 h3 8. Rc5+ Kg4 9. Nh2+ Kg3 10. Qg5+ Kf2 11. Qe5 hxg2# [All White moves were unique, so this is a real thematic try].

But 1. ... Ra8 ! refutes.

Solution: **1. h8=N+!** Kh5 2. g4+ hxg3 en passant 3. Qf7+ Kh4 4. Nf3++ Kg4 5. Ne5+ Kxg5 6. Nxd3+ Kg4 7. Qe6+ Kf3 8. O-O+ Rf2 9. Ne1+ Kf4 10. Ng2+ Kf3 11. Kh1 Rxf1#

This was the final version, in which short castling was turned into a try after a long and exhausting

process of refinement (and testing!). However, during the early stages of construction, castling was not a try at all: it was an actual cook. Even more remarkably, the obvious refutation 1...Ra8 did not yet exist. This produced a highly paradoxical situation. After the quiet move 1.O-O!, which appears strategically irrelevant and gives Black enormous freedom, every single Black reply among the sixteen legal possibilities failed to save him.

From a human perspective, such a move is exactly the type of continuation that one would be tempted to dismiss without extensive investigation. It neither creates an immediate threat nor forces Black into a narrow corridor of replies. On the contrary, it seems to relax the position and grant the defender additional options. Yet it was precisely this apparently harmless move that concealed a complete unintended solution. Hard to believe, isn't it? Just try by yourself.

This example illustrates one of the fundamental lessons repeatedly encountered during the verification of long selfmates. Human intuition is often a poor guide when judging the danger of secondary continuations. Moves that appear strategically absurd, aesthetically unnatural or

completely harmless may still hide fully functional solutions. The greater the length of the problem, the more opportunities exist for such hidden resources to emerge many moves later. For this reason, seemingly nonsensical continuations should never be discarded merely because they "look impossible". The logic of composition demands exactly the opposite approach: every legal continuation must be treated as potentially dangerous until it has been conclusively eliminated.

This was perhaps the most important philosophical motivation behind the entire verification effort. The objective was never simply to demonstrate that the intended solution works. It was to demonstrate that nothing else works.

And ultimately, to seal that certainty forever.

Final personal note

Before concluding this book, I wish to add a brief personal remark.

The observations, methodologies and technical considerations presented in these pages were not

developed by a computer scientist, a software engineer, or a professional programmer.

I have no degree in computer science, nor mathematics, and would not be capable of writing even a simple line of script in C, Pascal, Java, Python, PHP, or any other programming language. Everything described here was learned empirically during the verification itself, through direct observation, experimentation, trial and error, and countless hours spent studying the behaviour of Gustav under increasingly difficult conditions.

This detail is not mentioned as a merit, but as a curiosity.

In an age where highly specialized knowledge is often regarded as inaccessible without formal training, this project represents a small reminder that passion, persistence and genuine curiosity can still take a person surprisingly far into unfamiliar territory.

The purpose of this book was never to demonstrate technical expertise. It was simply to document a journey: a long attempt to answer a difficult question as rigorously as possible, using the tools that were available and learning whatever was necessary along

the way.

If the reader has found any value in these pages, perhaps the most important lesson is not about selfmates, brute-force verification, hash tables, or computer hardware. It is that determination can sometimes compensate for a remarkable amount of ignorance.

At least for a while.

Acknowledgements

I would also like to express my sincere gratitude to Olaf Jenkner, creator of Gustav in 1990.

Over the years, he has consistently and generously answered countless questions regarding the program, its internal mechanisms, its search strategies, and its practical use in the verification of selfmates and other direct-mate genres. He provided to send new versions when available. His explanations often provided valuable insights that would have been difficult or impossible to obtain otherwise.

Particular thanks are due for his patience. Many of these exchanges involved long lists of technical questions, sometimes several at once, yet detailed

and helpful answers were invariably provided. While any mistakes, interpretations, or speculative conclusions contained in this book remain entirely under the responsibility of myself, much of the understanding that made this work possible originated from those discussions.

For this reason, I wish to acknowledge not only the importance of Gustav itself, but also the remarkable availability of its creator in supporting the composition community over many years.

Of course, I also have to thank my old faithful Intel i3-4150, my parent's Intel i5-7400 (and my parents themselves, which did not protest for having their personal computer seized for weeks and remotely controlled every day), and my life-saver UPS power supply!

Daniele Guglielmo Gatti

FINAL VERDICT: The problem is Brute-Force C+.

Ending of the analysis: **July 2nd, 2026 – 6:00 A.M.**

Total time elapsed: **86 days.**

THE BORING PART

For those interested in details, here is a summarizing table.

Move or sequence	Purpose	Testing notes	Conclusions
1. g8=N+	Demonstrate insolubility.	Tested **1. ... Kg6**	No solution has been found. All the variations have been refuted. 1. g8=N+ is proved to be insoluble.
1. h8=R+	Demonstrate insolubility.	Forced reply **1. ... Kg6**	Needed to split moves – Computational intractability.
Split after **1. h8=R+ Kg6:**	Make the brute-force analysis feasible in reasonable time.	**1** excluded move for immediate checkmate. **54** tested moves.	No solution has been found. All the variations have been refuted. 1. h8=R+ is proved to be insoluble.

1. h8=Q+	Demonstrate uniqueness and correctness of the solution.	Forced reply **1. ... Kg6**	Needed to split moves – Computational intractability.
Split after **1. h8=Q+ Kg6:**	Make the brute-force analysis feasible in reasonable time.	**1** excluded move for immediate checkmate. **2** excluded moves for proved forced transposition to previously tested variations. **51** tested moves.	Proved that only 1. h8=Q+ leads to a solution. Proved that after 1. h8=Q+ Kg6, only 2. g8=R+ leads to a solution.
Analysis of the variation **1. h8=Q+ Kg6 2. g8=R+ Kxf7**	Demonstrate the absence of cooks after this forced variation, and that only 3. e8=B+ leads to solution. Demonstrate absence of duals in the 3.	**3** excluded moves for immediate checkmate. **54** tested moves. These moves were tested separately because of partial	Proved that only 3. e8=B+ leads to solution. Proved no duals after 3. e8=B+.

	e8=B+ variation, until the end of the problem.	computational intractability. *This was the best place for separate testing, due to the relatively limited amount of time needed to refute the majority of the moves (problem length was reduced by 2 moves, and every single test reduced of 1 further move).*	
Non-checking moves in initial position	Complete the brute-force analysis to be 100% sure of soundness.	**45** moves tested. Many splits (and splits of the splits) have been necessary.	No solution has been found. All the moves have been refuted.

EDITORIAL NOTE

This book is provided free of charge and may be freely distributed worldwide in its original form.

No modifications or alterations of any kind are permitted.

Copyright © 2026 Daniele Guglielmo Gatti. All rights reserved.

For contact:

danielegatti87@gmail.com