Solved Openings in Losing Chess

Mark Watkins,
School of Mathematics and Statistics, University of Sydney

1. INTRODUCTION

Losing Chess is a chess variant where each player tries to lose all one's pieces. As the naming of "Giveaway" variants has multiple schools of terminology, we state for definiteness that captures are compulsory (a player with multiple captures chooses which to make), a King can be captured like any other piece, Pawns can promote to Kings, and castling is not legal. There are competing rulesets for stalemate: International Rules give the win to the player on move, while FICS (Free Internet Chess Server) Rules gives the win to the player with fewer pieces (and a draw if equal). Gameplay under these rulesets is typically quite similar.[1] Unless otherwise stated, we consider the "joint" FICS/International Rules, where a stalemate is a draw unless it is won under both rulesets.

There does not seem to be a canonical place for information about Losing Chess. The ICGA webpage [H] has a number of references (notably [Li]) and is a reasonable historical source, though the page is quite old and some of the links are broken. Similarly, there exist a number of piecemeal Internet sites (I found the most useful ones to be [F1], [An], and [La]), but again some of these have not been touched in 5-10 years. Much of the information was either outdated or tangential to our aim of solving openings (in particular responses to 1. e3),

We started our work in late 2011. The long-term goal was to weakly solve the game, presumably by showing that 1. e3 wins for White. At that time, to the best of our knowledge there were 13 Black responses to 1. e3 that were proven to be losses (see §3). We proved that 1. e3 Nc6 (early 2012) and 1. e3 b5 (August 2012) are both won for White. When relaying this information to Frâncu (and others), he replied that he had recently shown 1. e3 Nh6 was a White win when testing a new laptop, and 2 weeks later we showed 1. e3 g5 is a White win. More recently (August 2014), we proved 1. e3 e6 is also a White win, leaving b6 and c5 as the remaining Black responses. We cannot say that our techniques show forth much innovation. Some of our results could presumably have been computed many years ago. Our contribution was mostly made possible by patience and hardware advances.

The author would like to thank Gian-Carlo Pascutto and Lenny Taelman for useful comments at an early stage of this work, Carl Lajeunesse [La] and Cătălin Frâncu [F1] for their websites, Klaas Steenhuis both for his enthusiasm for the project and concrete suggestions, Ben Nye for historical information, and both him and Ronald de Man for tablebase guidance. We have set up a webpage for downloading our programmes, including tools for search, verification, and tree manipulation, and the GUIs for controlling search and exploring final proof trees. The URL is: `http://magma.maths.usyd.edu.au/~watkins/LOSING_CHESS`

2. SEARCH METHODOLOGY

The forced-capture aspect of Losing Chess makes it well-suited for proof number (PN) search [Al]. Indeed, we could describe our search methods directly in terms of PN^2-search [BUH], but I personally think a slighltly different explication is more useful. Rather, we consider proof-number searches to be akin to an evaluation function. That is, for a given position P we search a certain amount of nodes (or time) in PN-search. The result of this search consists of the proof and disproof numbers for P, and similarly for all its (immediate) children. These are then stored in a higher-level tree,[2] and an evaluation for each leaf node is determined from these proof/disproof numbers, for instance as the ratio of the two (this is an idea of Frâncu),

[1] In contrast, the *vinciperdi* ruleset (which we do not consider here) has stalemate as a draw and games tend to take on a much different flavour, as Black's strategy is typically to block a lot of pawns, which greatly increases the difficulty of White to lose all the pieces.

[2] The PN-search results below the children of P are largely discarded. However, we also copied (recursively, starting from P) the information from any child node whose subtree (by a crude count) was at least (say) 70% as large as that of its parent. One nicety about this is that a highly forced line can produce many upper-level nodes from one PN-search, rather than needing multiple PN-searches.

The "opening book" trees at the websites [F1] or [La] work similarly. However, it appears that our PN-searches are deeper than theirs. For instance, [F1] lists a position as "trivial" if it can be solved (by Nilatac) in a 2 million node search (about 2 seconds), while our typical baseline was 10^7 nodes (or 10 seconds)[3] at the beginning of our project, or 10^8 nodes (100s) when solving 1. e3 e6. Our termination criteria at leaf nodes inside the PN-search were: side to play has no move, only 4 units left (tablebases), repetition of position, and opposite Bishop draw-or-better for Black (see below). Our initialisation of proof/disproof numbers followed the "mobility" weighting, namely being respectively set as 1 and the number of legal moves.

We then allowed the upper-level tree to grow, choosing leaf nodes to expand via PN-search in some best-first manner, such as minimaxing the leaf ratios/evaluations up to the root and expanding a critical leaf. To introduce some randomness into the selection process, we in essence took an idea from how opening books randomise their choices. We walked down the upper-level tree from the root node, at each step choosing a child randomly according to a weighting from its minimaxed proof/disproof ratio (compare [ST]).[4]

For transpositions, in the PN-search we chose to identify nodes corresponding to the same position only when the reversible move count was zero.[5] This had the advantage of dispelling any loops, though of course it is not very optimal. In the upper-level tree we (in the most recent versions) also identified nodes which were known wins for White. In the node counts given below, we enumerate each unique position only once.[6]

The above formed the basis of our automated search process. Originally it ran on a 6-processor machine (so six upper-level nodes were being expanded at any given time), and produced about 1 million upper-level nodes in an overnight run. Later, when solving 1. e3 e6, we switched to a 176-core cluster (Footnote 3 has more hardware details). After a bit of initial experimentation, it was found to be quite advantageous to declare a draw to be a win for Black. This had the effect of clearing up a lot of repetition draws from the upper-level trees. Another benefit was that positions where Black had a lone Bishop and White had one of the opposite colour (amongst other material) could be deemed Black wins. Tablebases (described below) were of course also quite powerful.

One disadvantage of the ratio-expansion is that one can sometimes wander into a "well" where almost all White moves have a great advantage, but none easily lead to wins. This is typical when White has a large material advantage (thus big mobility edge) and Black a lone king, but White needs to push a pawn (or two) to promotion before the final wipeout. Another quirk is that there can be a rather notable tempo-advantage in ratio-expansion.

We briefly mention possible improvements. One idea that we never implemented (at either the upper-level or in the PN-search) was a killer heuristic at sibling nodes; for instance, one could change the move priorities via modifying the evaluation from the ratios. Another idea (compare enhanced transposition cutoffs [ETC]): upon creation of a upper-level node, look at its possible children, and see if any (via transposition) are already known to give a result that proves the node. A similar task could be to avoid dominated lines. A final consideration is that the predictive value of proof/disproof ratio seems related to game phase, that is, a ratio of 100.0 with many pieces left on the board will often be solvable rather quickly, but the same ratio in a situation where Black has only a King and pawns is rather likely to just be a slow endgame win.

The above search procedure was augmented by human input in two main ways. Our upper-level trees could grew so large (tens of millions of nodes) so as to be in multiple files (typically corresponding to different tries for White), and thus there was some human choice is choosing which line to examine next. The second enhancement was a Java-based GUI interface[7] (inspired by the website of Carl Lajeunesse [La]) that allowed the user (namely myself) to choose what upper-level node to expand. This could either be done with leaf nodes directly, or by moving the root of the randomisation walk to a place of interest. This helped overcome various problems where

[3] We have used various hardware configurations, most commonly Phenom 1065T (2.95Ghz) originally, and then Xeon E5-2670 (2.6GHz). It is my feeling (though have no data to back this up) that larger PN-searches are superior, with the choices of 10^7 and 10^8 being RAM-limited.

[4] This also allows a cheap parallelisation by running independent PN-search instances on randomly chosen leaf nodes, updating the upper-level tree when a new result arrives, and then immediately starting a new leaf expansion via a random walk. Indeed, the website [La] operates similarly. However, the efficacy of this parallelisation method is debatable; it works reasonably well when there are enough useful leaf nodes to expand, which was typically true for us except at the very beginning or very end of a proof. As an estimate, our upper-level trees could have (tens of) millions of nodes of which thousands would be of relevance to a solution, while we used at most around 200 cores.

[5] Presumably one could be more highbrow in a couple of standard situations, namely forced *en passant* (such as both g3 and g4 having only hxg3 as a response) and multiple promotions when the promoting unit must then be captured.

[6] It might be preferable to enumerate the arcs of the proof graph (labelled by moves), for these are effectively what the data structure stores. To give an idea of the difference, the final proof tree for 1. e3 b5 had 82084261 nodes and 3860735 additional arcs corresponding to transpositions. Of the nodes themselves, 75657405 were internal (the large percentage being typical of Losing Chess, for long chains of single-child nodes come about from forced moves), and thus 6426856 were terminal, with 1415338 of the latter being in 4-unit tablebases.

[7] This was called LosingGUI, but is now ClusterGUI. The Java code itself is adapted from the "ComradesGUI" of the IPPOLIT developers.

it seemed that the automatic methods were simply increasing the tree size without making significant progress. As a rough estimate, the solving of 1. e3 b5 took about 2-3 cpu-years, about 2-3 human work-months (some of this rather passive) using the Java-based interface, and about the same amount of human time in code development, including learning enough about Losing Chess and previous programmes so as to have an idea of how to proceed.[8]

The desired result of the above process would be a (pruned) upper-level tree which was completely solved. One then still needs to expand this into a full proof tree. Our current code simply saves a proof tree for any PN-search that is won for White, and then when the upper-level tree is complete these are read back to assemble the final proof tree.[9] This is then verified, checking a number of tree properties: that all Black moves are considered, that transposition-identified nodes are indeed the same position, that terminal nodes are White wins (either via stalemate or 4-unit TBs), and more. This found a number of problems at various stages of our work.[10] Frâncu has been able to transfer our proof of 1. e3 Nc6 into FICS rules with Nilatac, giving another partial verification of our work. The Java-based LosingGUI mentioned above was also adapted into a WinningGUI, that allows one to walk through a proof tree, also giving counts on the size of subtrees.

The efficacy of having at least some tablebases can be seen from the position with (say) a White King on d8 and a Black pawn on d5. This is a draw (Black will King the pawn), though the proof/disproof ratio from a PN-search of 10^7 nodes is around 300, as White has much more mobility (hence more options) than Black, at least at first. Tablebases are of course not novel for Losing Chess, though I could not find anything particular to International Rules that had been done.[11] We decided to adapt the RobboBases of IPPOLIT developer "Roberto Pescatore" (this seems to be a pseudonym).[12] After nontrivial modifications, the code built the 4-unit TBs in around 2 hours, and the 5-unit TBs in a couple of weeks. In solving 1. e3 e6 we built a few relevant 6-unit TBs. Following the RobboBases, we used distance-to-conversion (DTC) as the metric. In the PN-search, only the 4-unit TBs were accessed, and these were read from a flattened array of 2 bits per entry. This takes about 800MB of memory, and allows fairly fast access. The 5-unit TBs could presumably be accessed in the PN-search, at least near the root, via a compression scheme such as that described in [TB]. For normal chess, this reduces the size of the 5-unit TBs to about 450MB (both the Shredderbases and the RobboTripleBases are about this size, as are the recent SyzygyBases of Ronald de Man), at the cost of some additional computational overhead in capture resolution. Our upper-level trees would modify the evaluation/ratio when a 5-unit tablebase was reached, but as a goal of the project was to provide final proof trees which needed only the 4-unit TBs, we still expanded the relevant subtrees.

3. SOLVED RESPONSES TO 1. E3

Black has 20 responses to 1. e3, with 12 of these being fairly easy to refute. Two of them, namely d5 and d6, are particularly trivial. All of these are folklore, having been known for some time. In Table 1, we give the size of the proof trees (or graphs) we obtained (the sizes can easily vary by 50% in independent solvings).

	nodes		nodes		nodes		nodes		nodes		nodes
b5	82084261	Nh6	17488664	f6	510661	f5	90297	h6	48591	Na6	3309
g5	45546357	Nc6	10755947	a5	353391	h5	69132	Nf6	22838	d6	33
e6	43471711	c6	2135914	a6	243154	e5	43276	g6	4489	d5	33

Table 1: Sizes of our proof trees for responses to 1. e3

1. e3 c6 (Andryushkov Defence, according to Anderjić)
This was first solved by Ben Nye's program ASCP in February 2003 (see [An]), and a solution also exists in Nilatac's opening book [F1]. We largely copied over Nilatac's tree manually, but also found a few simplifications.

[8] It seems very difficult to speculate how much human effort could (in retrospect) be automated. In the last 2 years I learned a lot about Losing Chess and which types of positions are likely to be (quickly) solvable, but forming specific (testable) heuristics is really a different project. To reiterate: my goal was to try to solve Losing Chess (using whatever means available), as opposed to researching PN-search.

[9] As an example, solving 1. e3 b5 generated approximately 50 million upper-level nodes (spread across many files), each corresponding to a PN-search of around 10^7 nodes. The pruned upper-level tree had only about 300000 nodes, and the full proof tree around 82 million nodes.

[10] The major disadvantage from an independence standpoint is that it uses the same move generation and tablebases as the main programme. Much of our move generation code followed that of IvanHoe (from the IPPOLIT developers), adapted suitably for Losing Chess.

[11] Ben Nye built 5-unit TBs for FICS Rules over a decade ago (see [B]. More recently Ronald de Man has built a number of 6-unit TBs.

[12] In the end, we used almost none of the clever ideas it contained. For instance, their (fast) index-differencing depends on the king-structure of normal chess, so we instead chose to always re-compute the index from scratch. Similarly, their SMP-slicing exploited the king structure, and with this not directly available we just ran it on one core. We also omitted the idea of a BlockedPawn counting as one unit.

1. e3 Nc6 (Balkan Defence)
From what I know, this was not previously proven to be a loss, but it is not *that* much harder than 1. e3 c6.

1. e3 b5 (Classical Defence)
Again the previous status is unclear to me. Pascutto seemed to remember seeing a lecture about its resolution (or something related) some years ago. We can note in passing that the "Suicide Defence", namely 1. e3 b5 2. Bxb5 Bb7, has long been known to be losing for Black.

1. e3 Nh6 (Hippopotamus Defence)
Of the 7 unsolved lines from when this project began, Nilatac's book gave this one the highest proof/disproof ratio (that is, most likely to be a win). When we announced our results on 1. e3 b5 privately (August 2012), Cătălin Frâncu responded that he had recently shown this line was a White win under FICS rules, taking about two cpu-months of computing time. We instrumented a utility to transfer his upper-level tree to our set-up. It took about 12 cpu-hours to try to solve all the nodes in his tree, leaving a handful of nodes that were solved with a few minutes of manual-driven work. This was then expanded into a full proof tree of 17.5 million nodes.

1. e3 g5 (Wild Boar Attack)
We then turned to 1. e3 g5, where 2. Ba6 bxa6 had been solved by Frâncu. Black's alternate try of 2. Ba6 Nxa6 took under a week to solve (about a cpu-month). Transferring Nilatac's 2. Ba6 bxa6 proof saw no problems, with about 5.7 million nodes in this subtree.

1. e3 e6 (Modern Defence)
Two years after the above results, we succeeded in solving 1. e3 e6. We used much more substantial hardware here, and eventually reduced the line to some 7-unit positions, most involving KRNPPP versus K. We then contacted Ronald de Man, who found that his solver (which uses 6-unit TBs in PN-search) was already able to solve our remaining positions (all our target positions turned out to be rather easily convertible 6-unit wins). We then built a few relevant TBs and solved the upper-level tree, obtaining a final proof tree of 43 million nodes. This is smaller than b5 and also g5, but was much harder to resolve, as White has many promising alternatives.

4. BIBLIOGRAPHY

[Al] L. V. Allis, M. van der Meulen, H. J. van den Herik, *Proof-number search*. Artificial Intelligence **66** (1994), 91–124. http://dx.doi.org/10.1016/0004-3702(94)90004-3. See also: L. V. Allis, *Searching for Solutions in Games and Artificial Intelligence*, Ph. D. thesis, Univ. Limburg, 1994.

[B] J. Beasley, *Losing Chess: Burning the Candle at Both Ends*. Variant Chess **41** (Jan 2003), page 8. http://www.mayhematics.com/v/vol6/vc41.pdf

[BUH] D. Breuker, J. Uiterwijk, H. J. van den Herik, *The PN^2-search algorithm*. In *Advances in Computer Games 9*, ed. H. J. van den Herik & B. Monien, Univ. Maastricht (2001), 115–132. ISBN 90-6216-5761.

[ETC] A. Plaat, J. Schaeffer, W. Pijls, A. de Bruin, *Exploiting Graph Properties of Game Trees.* 13th National Conference on Artificial Intelligence (AAAI-96), Vol. 1, 234–239, ISBN 978-0-262-51091-2.

See also: *Nearly Optimal Minimax Tree Search?*, Technical Report 94-19. Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada.

[TB] J. Schaeffer, Y. Bjornsson, N. Burch, R. Lake, P. Lu, S. Sutphen, *Building the Checkers 10-piece Endgame Databases*. In *Advances in Computer Games* **10** (2003), edited by J. van den Herik, H. Iida, E. A. Heinz, Advances in Information and Communication Technology Vol. 135, Kluwer, 193–210. ISBN 1402077092.

[ST] Y. Shoham, S. Toledo, *Parallel Randomized Best-First Minimax Search*. Artificial Intelligence, **137** (2002), 165–196. http://dx.doi.org/10.1016/S0004-3702(02)00195-9

[An] V. Andrejić, http://poincare.matf.bg.ac.rs/~andrew/suicide/index.html

[F1] C. Frâncu, Nilatac program, http://catalin.francu.com/nilatac

[H] G. Haworth, Losing Chess, http://ilk.uvt.nl/icga/games/losingchess

[La] C. Lajeunesse, Suicide Chess web page, http://suicidechess.ca

[Li] F. Liardet, Losing chess, http://www.pion.ch/Losing