Chess endgame compression via logic minimization

Dave Gomboc and Christian R. Shelton

University of California, Riverside CA 92521 USA
dave_gomboc@acm.org cshelton@cs.ucr.edu

Abstract. Chess endgame tables encode perfect information to inform heuristic search and permit error-free play once the root position is within them. We introduce a new approach to their minimization, and demonstrate better probe performance than the state-of-the-art.

Keywords: Chess · endgame tables · logic minimization

1 Introduction

Chess play has been studied for over a century, decades before computing science itself existed as a discipline [38,22,28,35,23,5,25,7]. Today, machines are substantially stronger Chess players than top human experts, and the same can be said regarding many other similar traditional human games.

1.1 Game-theoretic error-free endgame play

A Chess endgame table (EGT) is a precomputed, known-correct source of information about Chess endgame positions. The first Chess EGTs were computed by Ströhlein [33]; seven-piece EGTs were first computed by Zakharov et al. [37] on the Lomonosov supercomputer, using tens of tebibytes. Chess engines that reach such pre-tabulated positions from within their heuristic searches can propagate back an exact score, which can be used either to improve on-line game play or to improve the accuracy and efficiency of off-line reinforcement learning [29] via endgame table rescoring [20].

1.2 State-of-the-art Chess endgame data compression

Syzygy EGTs are predominantly used at present because they are more compact than any widely-available alternative, while also being acceptably efficient to query. Syzygy has coverage for positions where en passant is possible, but has no data for where some castling right exists. By design, Syzygy stores misleading values for positions containing legal captures that achieve better compression: correct querying of them requires the concomitant use of a capture-based quiescence search and minimaxing the resulting values. We use (an updated version of) Fathom [11] to provide these capabilities.

For each covered material balance, Syzygy includes both a win-draw-loss (WDL) table, and a distance-to-zeroing-move (DTZ) table. WDL alone is sufficient to avoid entering a game-theoretically-suboptimal position and to determine the result of a game, as has been previously achieved in Checkers [27], so we do not consider DTZ further.

1.3 Approach

A Chess EGT may be viewed as a partial function that maps Chess positions to a game-theoretic outcomes. We employ two-level logic minimization to represent this function in a compact form. In Section 2, we introduce two-level logic minimization, and explain how we encode Chess positions as logic bits. We discuss the experiments that we undertake in Section 3, and summarize our contributions in Section 4.

2 Two-level logic minimization

Consider a partial function $P : \{0,1\}^n \rightarrow \{0,1\}^m$. An equivalent total function $T : \{0,1\}^n \rightarrow \{0,1,X\}^m$ exists, where an output of X indicates that we do not care which truth value is assigned to that output. The straightforward tabular representation of T would always contain 2^n rows. For succinctness, we use a matrix representation $M : \{0,1,X\}^n \rightarrow \{0,1,X\}^m$ of P, where an input of X indicates that the row is applicable regardless of the instantiated truth value of that input. Thus, a single row of matrix M with k inputs set to X is equivalent to specifying the 2^k compatible rows of the tabular representation of T.

The union of the input vectors where any of the outputs is assigned to either 0, 1, or X is considered to be part of the ON-cover (or F, for function), the OFF-cover (or R, for reverse), or the DC-cover (or D, for don't care), respectively. Each such cover is the sum of clauses; each clause (or "cube") is the product of individual inputs.

Definition 1. Two-level logic minimization *is the task of, having been provided with some matrix M that is consistent with P, identifying a matrix M' that is also consistent with P whose covers of interest have minimum cardinality.*

We first discuss a few important algorithms from the electronic design automation (EDA) literature; see Coudert [8] for coverage of additional historically-important techniques. We then describe the mapping from Chess endgame table data to $\{0,1,X\}$-vectors and how to employ logic minimization in this context.

2.1 MINI

The MINI logic minimizer [19] introduced the heuristic approach of iteratively improving cover cardinality via repeated cube expansion and reduction.

Positional cube notation Similarly to the one-hot encodings used in machine learning, positional cube notation (PCN) maps each specific value of an input variable to a different bit, allowing for efficient cube operations to be performed via bitwise logical operators. A multiple-valued input variable over the domain {ant, bee, cat} could be mapped as: ant $\rightarrow$ `100`; bee $\rightarrow$ `010`; cat $\rightarrow$ `001`. In contrast to a one-hot encoding, `111` is also valid PCN, representing "do not care". For each binary input variable v, PCN reduces to a bit pair $\bar{v}v$: $0 \rightarrow$ `10`; $1 \rightarrow$ `01`; $X \rightarrow$ `11`.

Distance-one merging Hong *et al.* report using merging two cubes that disagree on only a single variable for "computational advantage", as in these three examples:

before	0X**0**1100 0X**1**1100	10**0**X010 10**X**X010	X0**1**1001 X0**X**1001
after	0X**X**1100	10**X**X010	X0**X**1001

MINI iterates over each such input variable once, updating the sorted ordering of M' prior to processing each variable to ensure that the clauses are ordered to permit all potential merges involving that variable via a linear scan through the product clauses.

Expansion Distance-one merging is a particular form of cube expansion, which is the process of enlarging a cube so that it (hopefully) includes as many as possible of the minimum product terms, or *minterms*, of M' that must be covered, while avoiding covering any product terms that must not be covered (the collection of which constitutes the *blocking cover*). Any cube that is completely covered by another may be discarded, thereby reducing the cardinality of M'.

Reduction Once expansion has occurred, many cubes that partially overlap may cover the same minterms. Cube reduction is the process of shrinking a cube while ensuring that it continues to cover all minterms not already covered by any other cube. This can allow the cube to later grow in a different direction for better overall minimization.

2.2 ESPRESSO

Along with the unate recursive paradigm, Brayton *et al.* [6] introduced the irredundancy operation.

Irredundancy While expansion alone can eliminate many cubes, it does not eliminate any cube that does not end up completely encompassed by a single other cube. The irredundancy pass within ESPRESSO's expansion-irredundancy-reduction main loop exists to prioritize the cardinality minimization of M' via the detection and removal of such cubes that are nonetheless redundant with respect to multiple other cubes in advance of performing any reduction that could cause an available opportunity for cube removal to be forfeited.

Distance-one merging Like MINI, the ESPRESSO implementation used does support the ability to apply distance-one merging across multiple variables of the ON-cover in sequence. Though this capability is not enabled by default, the `espresso(1)` manual page suggests its use.

2.3 Pupik

Pupik [14,15] is based on processing ternary trees that represent Boolean functions [12]. It repeatedly performs single-variable absorption and complementation to combine cubes. We observe that a single distance-one merge encompasses both of those

operations, and that the full procedure described in Fišer *et al.* [14] is *precisely equivalent* to performing distance-one merging over F. The asymptotic analysis performed therein naturally does not account for practical benefits of the matrix representation such as its memory locality and amenability to bit vector operations.

2.4 A simple position encoding scheme

The Chess position encoding we use retains the traditional top-level division of Chess endgame positions by their material balance to permit straightforward comparison with the substantial body of prior work. More importantly, the scheme selected is relatively *uninformed* about Chess. Not only do our input vectors contain no machine-learned features, they also fail to manually capture basic Chess notions such as whether the player to move is in check or has at least one legal move that can be played. We have stayed far away from using any sort of bitboard representation [1] that could cause logic minimization-based image processing techniques [4,10,26] to become applicable. No counters are used; even the specific material balance in use is not encoded. Furthermore, we make no application of concepts that might assist logic minimization itself such as multiple-valued variables or reflected binary (a.k.a. Gray) coding. By doing so, we hope to convince the reader of the generality of the compression technique.

The input to the logic minimizer contains a matrix description of the universe of discourse: `0 000010 000000 010000 001001 010` is an example row of T. Of the 25 input bits, the first is `1` iff Black is to move. Chess boards contain 2^6 potential piece locations, so a sextet is used to specify the placement of each piece. Pieces are recorded in ♔♕♖♗♘♙♚♛♜♝♞♟ order. The final triplet is a multi-valued variable indicating whether White wins, draws, or loses with best play, or that we do not care. Were we processing the ♔♘♚♟ table, the row above would be interpreted as follows: White is to move; the white king is on c8; the white knight is on a8; the black king is on a6; the black pawn is on b7; White has a draw with best play. The complete table T for each four-piece material balance contains 2^{25} rows.

Note the austere simplicity of this representation. Extensive efforts have been made to identify indexing schemes that include all legal positions for a material balance, but as few additional illegal positions as possible [18,34,17,21]. It is also common for multiple indexing order permutations to be attempted for each material balance: once it is determined which variant turns out to yield the smallest file size after a subsequent layer of block compression is applied, the necessary data required to select which scheme is to be used for decompression is recorded near the beginning of the file. Instead, we rely upon logic minimization to combine adjacent cubes with compatible outputs.

This representation also permits labelling large blocks of positions with the same output vector *a priori*. For example, all positions where a black pawn is on the eighth rank are illegal. We could specify that we do not care about any such positions within the ♔♘♚♟ table using a single matrix row: `X XXXXXX XXXXXX XXXXXX 000XXX XXX`. For simplicity, we do not currently represent either castling or en passant rights.

2.5 Method

We construct one matrix M per material balance, with each row, where each row represents a (possibly illegal) position and its associated game-theoretic outcome. We then employ logic minimization to construct a more compact matrix M'. Our procedure first repeatedly employs distance-one merging in combination with matrix re-sorting until no further reductions can be found: we call this iterated distance-one merging *compaction*. Then, one or more ESPRESSO operators (e.g., expansion) are applied.

To probe the game-theoretic value of a position, the query position is encoded as aforementioned. Then, the M' for the appropriate material balance is scanned linearly until a match is found. The output bits of the matching entry dictate the returned result.

3 Experimentation

We explore the trade-off between the minimization time and quality, then compare the resulting on-disk size and time to query endgame positions our method and Fathom.

3.1 Two- and three-piece endgame tables

Our first experiment manipulates three processing conditions while processing the two- and three-piece tables: whether compaction is or is not performed; whether a single expansion pass or the full ESPRESSO algorithm will be executed; whether or not canonicalization is used. This last condition is explained immediately below, followed by discussing the results of this first experiment.

Canonicalization Symmetries in Chess endgames (and in other puzzles and games) have long been exploited [24,2,16,30,31,32,9]. A simply example of symmetry exploitation is that the Syzygy (and our) EGTs do not include the ♔♚♕ material balance. When needed, the ♔♕♚ table is probed using the inverted position, and the result translated. Additional symmetries do exist, especially in pawnless endgames.

For each equivalence class of positions defined by the available symmetries for a material balance, we designate one in particular as the canonical representation for which WDL data is recorded. All other positions within the equivalence class are assigned exclusively to the DC-cover. The probing operation must then translate to the canonical position during querying.

Results We perform experiments on two- and three-piece endgames. Regardless of the processing condition under test, for each of the trivially-drawn material balances (♔♚, ♔♗♚, and ♔♘♚), the ON-cover consistently resolves to a single row with all inputs marked as don't care and the outputs indicating a drawn result. We are nonetheless able to observe striking performance differences with the remaining material balances (♔♕♚, ♔♖♚, and ♔♙♚).

Cumulatively, 194 602 clauses in F exist between the six endgame tables under consideration prior to compaction; versus just 29 920 clauses in F afterward. This early

6.5x reduction in cubes yields a significant processing time advantage for the immediately following expansions, and moreover, takes negligible time to perform. Accordingly, we always apply compaction in our experiments hereafter, and not just to F, but also to R and to D.

Canonicalization causes a large majority of positions in each endgame under consideration to be classified as don't care, which substantially increases the opportunities for cube expansion, and thus for ON- and OFF-cover cardinality minimization. We do acknowledge that the use of canonicalization effectively slips some slight amount of domain knowledge into our lossless compression algorithm. However, symmetry exploitation is not inherently Chess-specific, and this same (and even more) domain knowledge is also exploited by Syzygy EGTs, which serves as our standard for comparison. As with compaction, the benefit is so clear that we always apply this process hereafter.

Executing the complete ESPRESSO algorithm noticeably improves ON-cover cardinality versus performing only a single expansion pass, but the associated time penalty is also noticeable. Thus, we proceeded to further investigate this tradeoff with larger material balances.

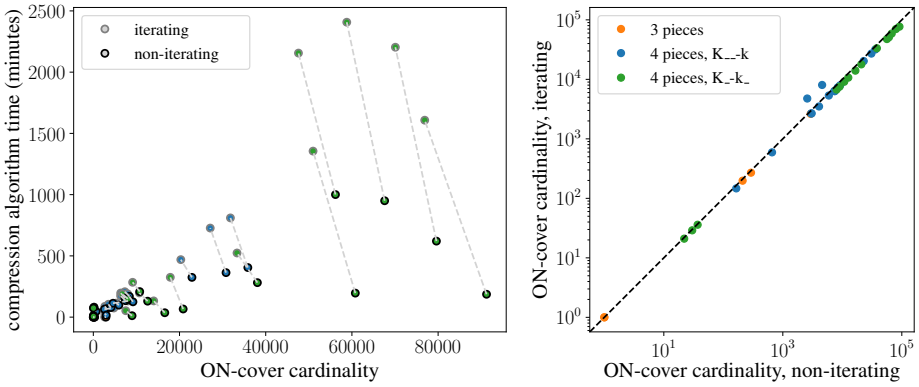


Fig. 1: (a) Time versus compression comparison for running ESPRESSO to completion versus performing just a single expansion and irredundancy pass. Each data pair, which is denoted by the dashed line connecting two dots, represents an endgame. The input to all endgames depicted was preprocessed using both compaction and canonicalization. (b) A comparison of the resulting cardinality of the ON-cover for the iterating versus the non-iterating treatment, using the same data as (a).

3.2 Two- through four-piece results

We now additionally include four-piece endgames in our exploration. The processing treatment that has been added is to apply compaction, expansion, and irredundancy, but without subsequent iteration of the ESPRESSO reduce-expand-irredundant main loop.

We observe that applying a single irredundancy pass after the expansion pass has been performed is both quick and effective at reducing the cardinality of F. Figure 1(a) illustrates that iterating the main loop of ESPRESSO extracts improvements only at a relatively high cost in time; Figure 1(b) provides an alternate view showing clearly that the ON-cover cardinality reduction achieved for this extra effort is not great. Furthermore, there is every reason to expect that iteration time will increase with problem size.

Exerting additional effort to continue to reduce cover cardinality may be particularly valuable in the electronics manufacturing context. For example, simpler circuits are associated with using either fewer lookup tables (LUTs), or less die space and power. However, the extra effort of iterating until improvement can no longer be found does not appear to be an efficient use of our limited processing power. Given our intention to obtain usable compressed PCN versions of larger EGTs as economically as possible, applying compaction, expansion, then irredundancy appears to be our best trade-off.

3.3 Compression effectiveness

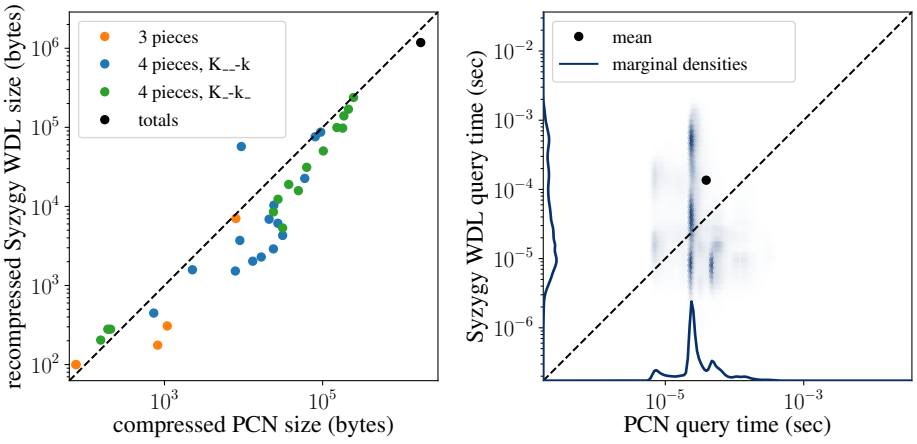
Each product clause in F generated when using compaction, expansion, and irredundancy is representable in PCN in 64 bits with room to spare. We generate a binary file per material balance containing each row in PCN sorted lexicographically, then compress it via `xz -9 -e`. Figure 2(a) contrasts the space consumed on disk by our method with the Syzygy endgame tables after being recompressed with `xz -9 -e`. For the 2-, 3-, and 4-piece endings, our method does require 47% more space than the recompressed Syzygy EGTs. Given that there has been a half-century of endgame table technology development leading to the Syzygy format, and that numerous opportunities to improve compression results using our novel method remain, we are comfortable claiming that this method of lossless compression has some promise.

3.4 Query effectiveness

We sampled uniformly with replacement to obtain ten million four-piece Chess endgame positions. We place a white king on any of 64 squares, then a black king on any of 64 squares, then any non-king on any of 64 squares, then any non-king on any of 64 squares, then select the side to move. Any non-legal position is then discarded. Note that castling and en passant rights are never present.

All ten million positions are first probed to verify correctness of the result returned. Afterwards, all ten million positions are probed a second time for timings capture. Fathom is used to probe Syzygy; our method linearly scans the minimized ON-cover for a matching cube.

Figure 2(b) shows the density plot of the joint distribution of query times. Densities above the diagonal are positions that took longer for Syzygy; those below took



(a) (b)

Fig. 2: Comparison of our method versus Syzygy WDL. (a) Table sizes: Each point represents one material balance; points above the diagonal line represent tables for which our method outperforms Syzygy. Note the log-scale: tables to the left and top dominate the total compression size. (b) Query performance: Joint density (across 10M random queries) of the query time for our method and Syzygy. Again, note the log-scales. Our method has a faster mean query time (39 µs versus 138 µs) and smaller standard deviation (41 µs, compared with 258 µs).

longer for our method. The mean time (above the diagonal) is shown, along with the marginal densities. Our probes are on average both substantially faster (39 µs versus 138 µs) and exhibit lower variablity (a standard deviation of 41 µs versus 258 µs), which demonstrates the viablity of the approach.

4 Contributions

Logic minimization techniques have previously been applied widely within EDA, and also within image processing-like and stream compression contexts [36,3]. We provide a top-level explanation of two-level logic minimization, clarify some relationships between techniques described within its literature, and demonstrate superior probing performance when using this method to losslessly compress Chess endgame tables.

Acknowledgments

We thank the University of California, Riverside for access to computational resources. The first author is also an employee of Google LLC, however, this research has been performed with neither awareness of any applicable insider Google LLC knowledge that might exist nor use of resources associated with that employment. Any statements and

opinions expressed do not necessarily reflect the position of the policy of either Google LLC or the University of California; no official endorsement should be inferred.

References

1. Adel'son-Vel'skii, G.M., Arlazarov, V.L., Bitman, A.R., Zhivotovskii, A.A., Uskov, A.V.: Programming a computer to play Chess. Russian Mathematical Surveys **25**(2), 221–262 (April 1970). https://doi.org/10.1070/rm1970v025n02abeh003792
2. Allis, L., van der Meulen, M., van den Herik, H.: Databases in Awari. In: Levy, D., Beal, D. (eds.) Heuristic Programming in Artificial Intelligence 2: the Second Computer Olympiad. pp. 73–86. Ellis Horwood (1991)
3. Amarú, L., Gaillardon, P., Burg, A., De Micheli, G.: Data compression via logic synthesis. In: Nineteenth Asia and South Pacific Design Automation Conference (ASP-DAC). pp. 628–633 (2014)
4. Augustine, J., Feng, W., Makur, A., Jacob, J.: Switching theoretic approach to image compression. Signal Processing **44**(2), 243 – 246 (1995). https://doi.org/https://doi.org/10.1016/0165-1684(95)00047-H
5. Bellman, R.: On the application of dynamic programing to the determination of optimal play in Chess and Checkers. Proceedings of the National Academy of Sciences of the United States of America **53**(2), 244 (1965)
6. Brayton, R.K., Sangiovanni-Vincentelli, A.L., McMullen, C.T., Hachtel, G.D.: Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers, Norwell, MA, USA (1984)
7. Campbell, M., Hoane, Jr., A.J., Hsu, F.h.: Deep Blue. Artif. Intell. **134**(1-2), 57–83 (Jan 2002). https://doi.org/10.1016/S0004-3702(01)00129-1
8. Coudert, O.: Two-level logic minimization: an overview. Integration, the VLSI Journal **17**(2), 97–140 (Oct 1994). https://doi.org/10.1016/0167-9260(94)00007-7
9. Culberson, J.C., Schaeffer, J.: Pattern databases. Computational Intelligence **14**(3), 318–334 (1998). https://doi.org/10.1111/0824-7935.00065
10. Damodare, R.P., Augustine, J., Jacob, J.: Lossless and lossy image compression using Boolean function minimization. Sadhana **21**(1), 55–64 (February 1996). https://doi.org/10.1007/BF02781787
11. Falsinelli, B., Dart, J., de Man, R.: Fathom. https://github.com/jdart1/Fathom (2015)
12. Fišer, P., Hlavička, J.: Implicant expansion methods used in the BOOM minimizer (2001)
13. Fišer, P., Kubátová, H.: Flexible two-level Boolean minimizer BOOM-II and its applications. In: 9th EUROMICRO Conference on Digital System Design (DSD'06). pp. 369–376 (August 2006). https://doi.org/10.1109/DSD.2006.53
14. Fišer, P., Rucký, P., Váňová, I.: Fast Boolean minimizer for completely specified functions. pp. 122–127 (2008)
15. Fišer, P., Toman, D.: A fast SOP minimizer for logic functions described by many product terms. pp. 757–764 (2009)
16. Gasser, R.: Solving Nine Men's Morris. Computational Intelligence **12**(1), 24–41 (1996). https://doi.org/10.1111/j.1467-8640.1996.tb00251.x
17. Heinz, E.: Endgame databases and efficient index schemes for Chess. Journal of the International Computer Games Association **22**(1), 22–32 (1999)
18. van den Herik, H., Herschberg, I.: The construction of an omniscient endgame data base. The Journal of the International Computer Chess Association **8**(2), 66–87 (1985)
19. Hong, S.J., Cain, R.G., Ostapko, D.L.: MINI: a heuristic approach for logic minimization. IBM Journal of Research and Development **18**(5), 443–458 (September 1974). https://doi.org/10.1147/rd.185.0443

20. killrducky: TB rescoring. https://blog.lczero.org/2018/09/tb-rescoring.html (2018)
21. Nalimov, E.V., Haworth, G.M., Heinz, E.A.: Space-efficient indexing of Chess endgame tables. Journal of the International Computer Games Association **23**(3), 148–162 (2000)
22. v. Neumann, J.: Zur theorie der gesellschaftsspiele ("On the theory of parlor games"). Mathematische Annalen ("Mathematical Annals") **100**, 295–320 (1928)
23. Newell, A., Shaw, J.C., Simon, H.A.: Chess-playing programs and the problem of complexity. IBM Journal of Research and Development **2**(4), 320–335 (1958). https://doi.org/10.1147/rd.24.0320
24. Patashnik, O.: Qubic: 4x4x4 tic-tac-toe. Mathematics Magazine **53**(4), 202–216 (1980). https://doi.org/10.1080/0025570X.1980.11976855, https://doi.org/10.1080/0025570X.1980.11976855
25. Quinlan, J.R.: Learning efficient classification procedures and their application to Chess end games. In: Machine Learning: an Artificial Intelligence Approach. pp. 463–482 (1983)
26. Sarkar, D.: Boolean function-based approach for encoding of binary images. Pattern Recognition Letters **17**(8), 839 – 848 (1996). https://doi.org/https://doi.org/10.1016/0167-8655(96)00045-1
27. Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., Sutphen, S.: Checkers is solved. Science **317**(5844), 1518–1522 (2007). https://doi.org/10.1126/science.1144079, https://science.sciencemag.org/content/317/5844/1518
28. Shannon, C.E.: A Chess-playing machine. Scientific American **182**(2), 48–51 (Feb 1950). https://doi.org/https://doi.org/10.1038/scientificamerican0250-48
29. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. Science **362**(6419), 1140–1144 (2018). https://doi.org/10.1126/science.aar6404
30. Stiller, L.: Parallel analysis of certain endgames. The Journal of the International Computer Chess Association **12**(2), 55–64 (1989)
31. Stiller, L.: Group graphs and computational symmetry on massively parallel architecture. The Journal of Supercomputing **5**(2-3), 99–117 (1991)
32. Stiller, L.: Multilinear algebra and Chess endgames. Games of no chance **29**, 151–192 (1996)
33. Ströhlein, T.: Untersuchungen über Kombinatorische Spiele ("Investigations on Combinatorial Games"). Ph.D. thesis, Fakultät für Allgemeine Wissenschaften der Technische Hochschule München ("Faculty of General Sciences of Munich Technical University") (1970)
34. Thompson, K.: Retrograde analysis of certain endgames. The Journal of the International Computer Chess Association **9**(3), 131–139 (1986)
35. Turing, A.M.: Digital computers applied to games. Faster than Thought (1953)
36. Yang, J., Savari, S.A., Mencercv, O.: Lossless compression using two-level and multilevel boolean minimization. In: 2006 IEEE Workshop on Signal Processing Systems Design and Implementation. pp. 148–152 (2006)
37. Zakharov, V., Maknhychev, V.: Creating tables of Chess 7-piece endgames on the Lomonosov supercomputer. Superkomp'yutery ("Supercomputers") **15** (2013)
38. Zermelo, E.: Über eine anwendung der mengenlehre auf die theorie des schachspiels ("On an application of set theory to the theory of the game of Chess"). Proceedings of the Fifth International Congress of Mathematicians **2**, 501–504 (1913)