Lossless compression via two-level logic minimization: a case study using Chess endgame data

Dave Gomboc **Christian R. Shelton**
Dept. of Computer Science and Engineering,
University of California, Riverside, CA 92521 USA
dave_gomboc@acm.org cshelton@cs.ucr.edu

Abstract

The utililty of processing techniques long in use within the electronic design automation community is underappreciated within the artificial intelligence community. We update and use the ESPRESSO logic minimizer in order to generate an exact, readily-queryable, and succinct representation of voluminous Chess endgame data.

1 Introduction

Chess play has been a relevant topic of academic study for over a century, decades before computing science itself existed as a discipline [Zermelo, 1913; v. Neumann, 1928; Shannon, 1950; Turing, 1953; Newell *et al.*, 1958; Bellman, 1965; Quinlan, 1983; Campbell *et al.*, 2002]. Today, machines are substantially stronger Chess players than top human experts, and the same can be said regarding many other similar traditional human games (e.g., Backgammon, Checkers, Go, Othello, Scrabble, Shogi, and Texas Hold 'em). Herein, we describe our recent effort to reduce the perceived infeasibility of creating a Chess program that, under the assumption that Chess is drawn with best play, will never make a move that leads to a lost position.

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 [1970]; seven-piece EGTs were first computed by Zakharov et al. [2013] 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 [Silver *et al.*, 2018] via tablebase rescoring [killrducky, 2018].

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

Syzygy endgame tables are the current standard Chess EGT format used, because these tables are more compact than any

Piece count	Required space
≤ 5	78.1 MiB
= 6	67.8 GiB
= 7	8.5 TiB

Table 1: Syzygy win-draw-loss endgame table sizes

widely-available alternative, while also being acceptably efficient to query. By their design, it is required that the probing software possesses considerable infrastructure of a Chess engine: in particular, positions with legal captures may be recorded using a misleading value that achieves better compression. The querying program must actually perform a complete capture-based quiescence search and minimax the resulting values in order to determine the correct result.

The open-source program Fathom [Falsinelli *et al.*, 2015] assists in querying Syzygy EGTs in the absence of a Chess engine. We used a locally-modified version of this program to query existing Syzygy tables to obtain source data. Accordingly, our computations have the same limitations as the Syzygy EGTs: for example, it is always assumed that neither side may castle their king.

For each covered combination of pieces, the Syzygy format provides a win-draw-loss ("WDL") table and a distance-to-zeroing-move ("DTZ") table. The ability to search from the starting position until the game result of every position in the search frontier can be looked up within the WDL tables alone would be sufficient for a player to never play into a game-theoretically-suboptimal position, as has previously been achieved in Checkers [Schaeffer *et al.*, 2007]: the DTZ tables can merely be helpful when exploiting any mistakes made by a fallible opponent. Accordingly, we will not consider the DTZ tables further herein.

1.3 Objectives

The exponential growth of tablebase size as the piece count increases ensures that the storage of even eight-piece tables on commodity hardware will not be feasible in the near future in the absence of an improved compression algorithm. We wish to sharply reduce the storage space required to represent Chess EGT WDL information, while also eliminating any requirement to perform any amount of game-tree search in order to successfully probe. We also hope that realizing substantial simplification or improvement in an already thoroughly-studied domain will encourage other scientists and engineers to consider whether processing the data related to their applications of interest in a similar manner would be beneficial.

We provide a fresh, accessible introduction to two-level logic minimization in Section 2, and report on our experiments in Section 3.

2 Two-level logic minimization

Let us 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. Naturally, the straightforward tabular representation of T would always contain 2^n rows. Succinctness is desirable, so the matrix representation of the function $M : \{0,1,X\}^n \rightarrow \{0,1,X\}^m$, where an input of X indicates that the row is applicable regardless of the instantiated truth value of that input, can be used to reduce the number of rows in the tabular representation: 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", by tradition, though a clause can represent a hyperrectangle) is the product of individual inputs. *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, though we refer the reader to Coudert [1994] for coverage of additional historically-important logic minimization techniques. We then describe the mapping from Chess endgame table data to $\{0,1,X\}$-vectors, and we finish the section by discussing the enhancements that we have made to ESPRESSO.

2.1 MINI

The MINI logic minimizer [Hong *et al.*, 1974] introduced the heuristic approach of iteratively improving cover cardinality via repeated cube expansion and reduction.

Positional cube notation

In positional cube notation ("PCN"), each specific value of an input variable v is mapped to a tuple of bits whose length is the cardinality of the domain. So, a multiple-valued input variable of the domain {ant, bee, cat, dog} could be mapped as follows: ant → `1000`; bee → `0100`; cat → `0010`; dog → `0001`. Here, `1111` would be used to represent "don't 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`. Cube intersection is efficiently performed via bitwise and: when two cubes with incompatible variable assignments are so intersected, a zero-tuple occurs for each such conflicting variable.

Distance-one merging

In their paper, Hong *et al.* report that MINI performed distance-one merging for "computational advantage". An individual distance-one merge operation permits two product clauses of a cover to be combined when they disagree in a single $\{0,1,X\}$-input variable, thereby reducing cover cardinality, as in the following three examples:

before	`0X`**`0`**`1 100` `0X`**`1`**`1 100`	`10`**`0`**`X 010` `10`**`X`**`X 010`	`X0`**`1`**`1 001` `X0`**`X`**`1 001`
after	`0X`**`X`**`1 100`	`10`**`X`**`X 010`	`X0`**`X`**`1 001`

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*). An expanded cube may newly encompass one or more other cubes: when this happens, these other cubes are no longer necessary to retain in order to accurately represent P, and so are discarded.

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.

2.2 ESPRESSO

Brayton *et al.* [1984] famously introduced the unate recursive paradigm in their book, and their C implementation of ESPRESSO was open-sourced under a liberal licence, assisting its wide adoption. Unfortunately, that version has remained in relative statis for the past quarter-century.

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 neither alluded to anywhere in the book nor happens by default, the `espresso(1)` manual page suggests its use.

2.3 Pupik

The *Pupik* logic minimization algorithm [Fišer *et al.*, 2008; Fišer and Toman, 2009] hails from the same research group as the BOOM-II heuristic logic minimizer [Fišer and Kubátová, 2006]. Pupik is based on processing ternary trees [Fišer and Hlavička, 2001] that compactly represent Boolean functions. Pupik repeatedly performs single-variable absorption $(a + ab = a)$ and complementation $(ab + a\bar{b} = a)$ to combine adjacent cubes with identical outputs.

Unfortunately, the algorithmic and experimental performance analyses performed by Fišer *et al.* [2008] and Fišer and Toman [2009] consider neither the possibility of maintaining M' in sorted order nor the use of ESPRESSO's distance-one merging capability, respectively. In fact, exploiting single-variable absorption and complementation together is *exactly the same operation* as a single distance-one merge operation, and performing the full procedure described in Fišer *et al.* [2008] is *precisely equivalent* to distance-one merging over F.

Furthermore, asymptotic analysis is not the whole story: performing repeated accesses over a tree lacks the memory locality behaviour of comparing vectors that are juxtaposed in memory[1], and assessing each individual binary bit access within a table as a distinct operation is also unrealistic. Today, commodity processors provide registers that support operating on 256 or even 512 bits at a time: it is difficult to say how many tens of thousands of binary inputs might be required before performance actually increased from using a tree structure without either access to Pupik's source code or reimplementing it from scratch.[2]

2.4 A simple position encoding scheme

Many different ways to encode Chess positions for subsequent logic minimization exist: we have chosen ours with two major critiera in mind. We retain the traditional top-level division of Chess endgame positions by their material balance. Even though another strategy could be superior, making this choice permits straightforward comparison of our experimental results with what has been prior practice for a half-century.

More importantly, the position encoding scheme has been selected to be as *absolutely uninformed* about Chess as possible. 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 [Adel'son-Vel'skii *et al.*, 1970] that could cause logic minimization-based image processing techniques [Augustine *et al.*, 1995; Damodare *et al.*, 1996; Sarkar, 1996] 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 technique presented herein.

Directives at the top of an ESPRESSO input file indicate how many inputs and outputs will appear per matrix row, and can also indicate how many rows are present and which covers will be provided. Such directives are followed by a matrix description of the universe of discourse: `0 000010 010000 000000 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 six bits are used to specify the placement of each piece. We always record the locations of the white king and black king in bits 2-7 and 8-13, respectively; additional sextets are used to represent any additional pieces for the particular material balance in use. The final triplet are the three outputs, which indicate whether the side that is to move wins, draws, or loses. Were we processing the ♔♘♚♟ table, the row above would be interpreted as follows: White is to move; the white king is on c8; the black king is on a6; the white knight is on a8; the black pawn is on b7; White has a draw with best play. The complete table T for the ♔♘♚♟ material balance that we provide to ESPRESSO for minimization contains 2^{25} rows.

The austere simplicity of this representation is noteworthy. 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 [van den Herik and Herschberg, 1985; Thompson, 1986; Heinz, 1999; Nalimov *et al.*, 2000]. 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. By instead relying upon logic minimization to combine adjacent cubes with compatible outputs, we avoid considerable tedium.

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`. Thus, enhancing EGTs to also accommodate positions where castling rights have not been lost becomes straightforward[3]: add a binary input for each relevant castling status, and whenever one is enabled, set to don't care the three outputs for all positions when either the relevant king or rook that would be castled with has already moved.

2.5 Modifications of ESPRESSO

We have made several local improvements to ESPRESSO. Most of the changes we described are to ensure the correctness of and/or to simplify the implementation, though compaction itself provides a substantial performance improvement, as we shall see in Section 3.

[1] In practice, a suffix array implementation exhibits an approximately 5x performance advantage over an equivalent suffix tree implementation, for this very reason.

[2] We attempted to but did not succeed in establishing communication with the relevant research authors.

[3] Castling support would be useful: Chess studies presume the legality of castling, unless it can be proven that an encountered position could not have been reached without having forfeited the right to castle.

endgame material balance	invalid positions	acanonical positions marked as don't care	illegal positions	side to move wins with best play	side to move draws with best play	side to move loses with best play	inputs (`.i`)	outputs (`.o`)	rows in T (`.p`)
♔♚	128	0	840	0	7224	0	13	3	2^{13}
♔♚	128	7508	64	0	492	0	13	3	2^{13}
♔♕♚	24 320	0	131 516	144 508	23 048	200 896	19	3	2^{19}
♔♕♚	24 320	465 648	9469	9563	1739	13 549	19	3	2^{19}
♔♖♚	24 320	0	100 856	175 168	22 244	201 700	19	3	2^{19}
♔♖♚	24 320	465 648	7108	11 924	1727	13 561	19	3	2^{19}
♔♗♚	24 320	0	82 740	0	417 228	0	19	3	2^{19}
♔♗♚	24 320	465 648	6105	0	28 215	0	19	3	2^{19}
♔♘♚	24 320	0	70 528	0	429 440	0	19	3	2^{19}
♔♘♚	24 320	465 648	5260	0	29 060	0	19	3	2^{19}
♔♙♚	24 320	0	168 616	124 960	108 788	97 604	19	3	2^{19}
♔♙♚	24 320	278 256	136 694	28 938	34 807	21 273	19	3	2^{19}

Table 2: Two- and three-piece endgame table statistics, both without and with canonicalization

Compaction
We perform distance-one merging against all $\{0, 1, X\}$-inputs over each of F, D, and R, but using ESPRESSO's existing data structures. Compaction can occur in the presence of multiple-valued inputs and/or multiple binary outputs. Unlike MINI's and ESPRESSO's older distance-one merging capabilities, we do not cease to iterate after visiting each input variable once. Instead, we continue iterating until no further distance-one merges are available.

Function cover consistency
While a function's covers need to be self-consistent, ESPRESSO's checking has been stricter than is necessary. Self-consistency does require that the union of a function's ON-cover, OFF-cover, and DC-cover must be the universe. However, overlap between its ON-cover and its DC-cover is permitted. Likewise, overlap between its OFF-cover and its DC-cover is permitted. Consequently, overlap between a function's ON-cover and OFF-cover that are simultaneously undergoing minimization should actually be permitted, so long as the entirety of their overlap remains within the function's DC-cover. In other words, the intersection of the OFF-cover and the DC-cover should not be part of the blocking cover when operating on the ON-cover, and likewise, the intersection of the ON-cover and the DC-cover should not be part of the blocking cover when operating on the OFF-cover.

ESPRESSO does not retain either $F - D$ or $R - D$ in memory: making such an improvement would permit simpler covers to be identified whenever both covers could take advantage of flexibility provided by using the same don't cares. Currently, we compute these as part of our improved consistency checking when vetting cover information read in from a data file, as in the following operation.

Function consistency
Verifying that two matrices A and B actually represent the same partial function P is essential to ensure that operations on M' have been correctly performed. Therefore, we added the capability to check that all of the following conditions hold for two matrices read in from their corresponding data files:

- $D_A = D_B$
- $F_A - D_A = F_B - D_B$
- $R_A - D_A = R_B - D_B$

Technical debt repayment
In furtherance of our aim to make additional algorthmic and technical improvements, including but not limited to supporting multiprocessing and SIMD-enablement via the usage of in-memory compressed Boolean vectors [Lemire *et al.*, 2018], we: have sharply reduced ESPRESSO's use of not only global variables and structures, but also raw memory accesses; have upgraded all of ESPRESSO's code, which was a mixture of K&R C and ANSI C89/ISO C90, to C++17; now use the CMake build system, permitting parallel compilation.

3 Experimentation

We conduct experiments to explore the trade-off between minimization time and minimization quality. As can be discerned from Table 2, the two-piece table has 2^{13} possible positions (where, as explained above, a position not only encompasses the mapping of pieces to squares but also includes the side to move), while each three-piece table has 2^{19} possible positions. Any position where a piece would be superimposed on another is considered to be invalid. The notion of canonicalization referred to in the third column of Table 2 will be described below.

The machine that we used for all timed computations has an eight-core i7-9700 CPU that nominally runs at 3.0GHz (though the frequency adapts dynamically) and has 32 GiB of RAM. Each job was given the use of a single core and 4 GiB of RAM. Up to seven jobs were permitted to run simultaneously, in an attempt to avoid any resource oversubscription that might cause undesirable timing variability.

acanonical as don't care	material balance	T clauses in F	just expansion time (s)	just expansion clauses in F	full ESPRESSO time (s)	full ESPRESSO clauses in F	just compaction time (s)	just compaction clauses in F	compaction, then just expansion time (s)	compaction, then just expansion clauses in F	compaction, then full ESPRESSO time (s)	compaction, then full ESPRESSO clauses in F
false	♔♚	7224	0.1	1	0.1	1	0.1	300	0.1	1	0.2	1
true	♔♚	492	0.1	1	0.2	1	0.1	94	0.1	1	0.1	1
false	♔♕♚	368 452	21 319.9	1275	22 524.9	1123	91.7	19 024	470.1	1379	929.7	1118
true	♔♕♚	24 851	121.8	196	477.2	184	19.1	4586	30.3	235	48.1	187
false	♔♖♚	399 112	30 169.6	846	30 660.9	746	58.4	15 096	280.5	920	476.1	741
true	♔♖♚	27 212	177.1	161	426.6	155	19.1	4415	24.8	191	38.7	158
false	♔♗♚	417 228	119.0	1	120.3	1	124.5	12 984	124.8	1	125.0	1
true	♔♗♚	28 215	10.2	1	15.6	1	14.0	4326	14.1	1	14.3	1
false	♔♘♚	429 440	41.6	1	42.7	1	44.0	8188	44.1	1	44.3	1
true	♔♘♚	29 060	9.3	1	14.7	1	12.9	3528	13.0	1	13.2	1
false	♔♙♚	331 352	8032.3	5407	17 592.1	4632	122.6	30 570	521.0	6612	3914.7	4657
true	♔♙♚	85 018	1368.4	2198	8529.6	1992	36.0	12 971	118.9	2706	621.7	2004

Table 3: Two- and three-piece endgame table results, with and without canonicalization and distance-one merging

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 the full ESPRESSO algorithm[4] will be executed versus just a single expansion pass; 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, e.g., Patashnik [1980], Allis *et al.* [1991], Gasser [1996], Stiller [1989; 1991; 1996], Culberson and Schaeffer [1998]) have long been exploited. A simple example of symmetry exploitation is that the Syzygy EGTs do not include the ♔♚♛ material balance. When it is needed, the ♔♕♚ table will instead be probed using the reversed board, and the response received will be translated also. Additional symmetries do exist, especially in pawnless endgames.

For each equivalence class of positions defined by the available symmetries for a material balance[5], we can 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 determine the appropriate canonical position, read the WDL data, and, depending upon the particular canonicalization transformation made, potentially translate the read data back in order to return the appropriate probe result.

Results

Table 3 shows ON-cover minimization results for two- and three-piece endgames under a variety of processing conditions. The three trivial material balances that consistently resolve to the same (drawn) result without regard to any position features all resolve quickly with all reported processing methods. We can nonetheless observe some striking performance differences with the remaining three material balances.

Performing distance-one merging across each of F, D, and R prior to the first expansion pass of ESPRESSO consistently yields a clear and substantial processing time advantage. Accordingly, we always apply compaction 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. Having examined only three nondegenerate material balances so far, we will further investigate this tradeoff with additional material balances later in this section.

Canonicalization substantially increases the opportunities for minimizing the cardinalities of the ON- and OFF-covers. As with compaction, canonicalization yields a substantial minimization-time processing advantage; thus, we always apply it hereafter.

[4] A full discussion thereof would take us far astray: see Brayton *et al.* [1984] for details.

[5] One caveat is that we do not yet take advantage of an additional symmetry that exists when White and Black have the same material, which limits performance in the ♔♕♚♛ and ♔♖♚♜ cases.

material balance	T clauses in F	compaction time (h)	compaction clauses in F	compaction, then just expansion time (h)	compaction, then just expansion clauses in F	compaction, expansion, then irredundancy time (h)	compaction, expansion, then irredundancy clauses in F	compaction, then full ESPRESSO time (h)	compaction, then full ESPRESSO clauses in F
♔♚	492	0.000	94	0.000	1	0.000	1	0.000	1
♔♕♚	24 851	0.005	4586	0.008	235	0.009	204	0.013	187
♔♖♚	27 212	0.005	4415	0.007	191	0.007	170	0.011	158
♔♗♚	28 215	0.004	4326	0.004	1	0.004	1	0.004	1
♔♘♚	29 060	0.004	3528	0.004	1	0.004	1	0.004	1
♔♙♚	85 018	0.010	12 971	0.033	2706	0.035	2243	0.173	2004
♔♕♕♚	1 199 825	0.820	125 819	18.016	6257	18.229	4681	25.767	4089
♔♕♖♚	1 385 863	0.876	120 377	9.778	2097	9.848	1702	12.485	1492
♔♕♗♚	1 429 843	1.392	149 698	11.911	6820	12.153	5026	21.509	4252
♔♕♘♚	1 465 555	1.290	133 683	9.731	3789	9.859	2905	15.048	2500
♔♕♙♚	4 308 718	5.152	221 360	45.909	8100	46.080	6020	68.381	5183
♔♖♖♚	1 385 236	0.554	107 206	6.227	415	6.245	320	6.553	288
♔♖♗♚	1 554 111	1.354	149 201	8.644	4421	8.796	3214	14.252	2640
♔♖♘♚	1 589 978	1.260	132 801	8.884	2588	8.968	1955	11.983	1685
♔♖♙♚	4 653 226	4.292	197 993	34.201	6035	34.315	4444	44.218	3860
♔♗♗♚	1 481 141	2.328	149 839	4.072	2579	4.170	1874	5.526	1747
♔♗♘♚	1 644 205	1.904	165 696	18.767	9343	19.103	6956	35.726	6042
♔♗♙♚	4 841 590	12.420	313 838	148.817	27 711	149.388	21 321	287.409	18 686
♔♘♘♚	1 562 304	1.054	106 484	5.085	110	5.091	90	5.172	76
♔♘♙♚	4 974 103	13.666	292 502	149.774	30 974	150.271	24 052	323.855	21 335
♔♙♙♚	3 803 456	5.134	262 735	98.877	23 176	99.146	16 777	169.534	14 480
♔♕♚♛	1 120 431	4.629	250 785	19.446	32 145	20.915	21 019	135.959	17 641
♔♕♚♜	1 329 609	1.642	153 187	13.767	12 247	14.154	8396	32.541	7279
♔♕♚♝	1 413 932	2.108	204 852	22.151	12 951	22.745	7878	51.109	6798
♔♕♚♞	1 461 810	1.604	166 111	13.353	9539	13.746	5015	22.371	4215
♔♕♚♟	4 316 927	9.388	308 727	67.112	24 827	67.648	18 944	143.350	16 958
♔♖♚♜	1 349 331	2.884	200 785	13.695	17 138	14.415	11 151	60.862	9432
♔♖♚♝	1 553 387	2.456	191 334	13.675	13 094	14.203	9258	40.476	7995
♔♖♚♞	1 601 265	2.691	202 919	15.192	29 200	16.114	21 890	134.434	18 682
♔♖♚♟	4 669 822	16.430	387 566	140.715	57 060	141.867	45 850	604.759	40 943
♔♗♚♝	1 480 764	1.693	153 678	10.012	36	10.015	33	10.034	33
♔♗♚♞	1 661 198	1.079	108 600	6.472	22	6.474	20	6.485	20
♔♗♚♟	4 870 169	10.361	407 929	79.057	40 749	80.032	30 224	320.567	25 952
♔♘♚♞	1 568 855	1.183	124 679	6.280	21	6.282	20	6.295	20
♔♘♚♟	4 669 822	9.480	410 373	78.993	46 307	79.876	35 363	342.261	31 040
♔♙♚♟	3 720 494	25.243	580 870	111.430	133 103	assertion failure		assertion failure	

Table 4: ON-cover cardinality reduction when using don't care for acanonical placements

3.2 Two- through four-piece results

We now include four-piece endgames as we attempt to explore further the trade-off of minimization time versus the cardinality of F. The processing treatment that has been added to Table 4 is to apply compaction, expansion, and irredundancy, without performing the full ESPRESSO algorithm.

Attempting to run ESPRESSO when using the ♔♙♚♟ material balance results in an assertion failure being issued from within the irredundancy portion of ESPRESSO's code. The root cause is that the 16-bit field used to store cube indices while determining an irredundant cover is insufficiently capacious when processing sufficiently large data sets.

We assess the irredundancy pass to be both relatively quick and effective at further reducing the cardinality of F after the expansion pass has been performed. When the full ESPRESSO algorithm also completes relatively quickly, it is in cases where it provides no substantial additional minimization over and above what expansion and irredundancy together achieve. In many other cases, running the full ESPRESSO algorithm is extremely time-consuming, and furthermore, there is every reason to expect it to remain prohibitively expensive as problem size increases.

The additional effort to continue to reduce cover cardinality undertaken by the full ESPRESSO algorithm may be particularly valuable in the electronics manufacturing context. For example, simpler circuits are associated with using either fewer lookup tables ("LUT"s), or less die space and less power. However, the extra effort of executing full ESPRESSO 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

We now take each product clause in F generated when using compaction, expansion, and irredundancy. Each clause can be represented in PCN in 64 bits with room to spare. We have generated a binary file per material balance containing each row in PCN. Table 5 reports the bytes consumed by the uncompressed PCN in memory (which is eight bytes per product clause in F), the bytes consumed by the compressed PCN files that would be stored persistently, and the size of the (already compressed) Syzygy WDL tables that are currently stored persistently when used with Chess engines.

The final row of Table 5 shows that the WDL information for all successfully processed endings requires 948 292 bytes in the compressed PCN format, versus 1 017 456 bytes for the same data in the Syzygy WDL format. 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 this novel method remain, we are comfortable claiming that this method of lossless compression has promise.

4 Contributions

Logic minimization techniques have previously been applied widely within EDA, and also within image processing-like and stream compression contexts [Yang *et al.*, 2006; Amarú *et al.*, 2014]. We have provided a top-level explanation of essential two-level logic minimization algorithms, clarified some relationships between techniques described within its literature, and shown experimentally that logic minimization may be an effective technique for compressing 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 or the policy of either Google LLC or the University of California; no official endorsement should be inferred.

material balance	uncompressed PCN size (B)	compressed PCN size (B)	Syzygy WDL size (B)
♔♚	8	64	n/a (80)
♔♕♚	1632	748	272
♔♖♚	1360	636	208
♔♗♚	8	64	80
♔♘♚	8	64	80
♔♙♚	17 944	6252	7824
♔♕♕♚	37 448	13 652	7056
♔♕♖♚	13 616	5096	4560
♔♕♗♚	40 208	14 332	4944
♔♕♘♚	23 240	8728	3600
♔♕♙♚	48 160	18 432	12 496
♔♖♖♚	2560	1244	1936
♔♖♗♚	25 712	8976	2832
♔♖♘♚	15 640	5776	2320
♔♖♙♚	35 552	13 816	5136
♔♗♗♚	14 992	5748	58 000
♔♗♘♚	55 648	21 040	7632
♔♗♙♚	170 568	60 612	81 424
♔♘♘♚	720	424	1360
♔♘♙♚	192 416	69 272	93 200
♔♙♙♚	134 216	47 460	25 104
♔♕♚♛	168 152	66 140	16 528
♔♕♚♜	67 168	25 956	20 496
♔♕♚♝	63 024	23 160	6672
♔♕♚♞	40 120	15 820	10 064
♔♕♚♟	151 552	54 624	58 064
♔♖♚♜	89 208	36 328	12 944
♔♖♚♝	74 064	29 092	32 912
♔♖♚♞	175 120	69 512	100 048
♔♖♚♟	366 800	131 720	179 408
♔♗♚♝	264	176	1232
♔♗♚♞	160	140	2256
♔♗♚♟	241 792	88 784	107 472
♔♘♚♞	160	140	1168
♔♘♚♟	282 904	104 264	148 048
totals	2 552 144	948 292	1 017 456

Table 5: Endgame table sizes

References

[Adel'son-Vel'skii *et al.*, 1970] G. M. Adel'son-Vel'skii, V. L. Arlazarov, A. R. Bitman, A. A. Zhivotovskii, and A. V. Uskov. Programming a computer to play Chess. *Russian Mathematical Surveys*, 25(2):221–262, April 1970.

[Allis *et al.*, 1991] L.V. Allis, M. van der Meulen, and H.J. van den Herik. Databases in Awari. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2: the Second Computer Olympiad*, pages 73–86. Ellis Horwood, 1991.

[Amarú *et al.*, 2014] L. Amarú, P. Gaillardon, A. Burg, and G. De Micheli. Data compression via logic synthesis. In *Nineteenth Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 628–633, 2014.

[Augustine *et al.*, 1995] Jacob Augustine, Wen Feng, Anamitra Makur, and James Jacob. Switching theoretic approach to image compression. *Signal Processing*, 44(2):243 – 246, 1995.

[Bellman, 1965] Richard Bellman. 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.

[Brayton *et al.*, 1984] Robert King Brayton, Alberto L. Sangiovanni-Vincentelli, Curtis T. McMullen, and Gary D. Hachtel. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Norwell, MA, USA, 1984.

[Campbell *et al.*, 2002] Murray Campbell, A. Joseph Hoane, Jr., and Feng-hsiung Hsu. Deep Blue. *Artif. Intell.*, 134(1-2):57–83, January 2002.

[Coudert, 1994] Olivier Coudert. Two-level logic minimization: an overview. *Integration, the VLSI Journal*, 17(2):97–140, October 1994.

[Culberson and Schaeffer, 1998] Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[Damodare *et al.*, 1996] R. P. Damodare, J. Augustine, and J. Jacob. Lossless and lossy image compression using Boolean function minimization. *Sadhana*, 21(1):55–64, February 1996.

[Falsinelli *et al.*, 2015] Basil Falsinelli, Jon Dart, and Ronald de Man. Fathom. https://github.com/jdart1/Fathom, 2015.

[Fišer and Hlavička, 2001] Petr Fišer and Jan Hlavička. Implicant expansion methods used in the BOOM minimizer. 2001.

[Fišer and Kubátová, 2006] Petr Fišer and Hana Kubátová. Flexible two-level Boolean minimizer BOOM-II and its applications. In *9th EUROMICRO Conference on Digital System Design (DSD'06)*, pages 369–376, August 2006.

[Fišer and Toman, 2009] Petr Fišer and David Toman. A fast SOP minimizer for logic functions described by many product terms. pages 757–764, 2009.

[Fišer *et al.*, 2008] Petr Fišer, Přemysl Rucký, and Irena Váňová. Fast Boolean minimizer for completely specified functions. pages 122–127, 2008.

[Gasser, 1996] Ralph Gasser. Solving Nine Men's Morris. *Computational Intelligence*, 12(1):24–41, 1996.

[Heinz, 1999] E.A. Heinz. Endgame databases and efficient index schemes for Chess. *Journal of the International Computer Games Association*, 22(1):22–32, 1999.

[Hong *et al.*, 1974] S. J. Hong, R. G. Cain, and D. L. Ostapko. MINI: a heuristic approach for logic minimization. *IBM Journal of Research and Development*, 18(5):443–458, September 1974.

[killrducky, 2018] killrducky. TB rescoring. https://blog.lczero.org/2018/09/tb-rescoring.html, 2018.

[Lemire *et al.*, 2018] Daniel Lemire, Owen Kaser, Nathan Kurz, Luca Deri, Chris O'Hara, François Saint-Jacques, and Gregory Ssi-Yan-Kai. Roaring bitmaps: Implementation of an optimized software library. *Software: Practice and Experience*, 48(4):867–895, 2018.

[Nalimov *et al.*, 2000] E. V. Nalimov, G. McC. Haworth, and E. A. Heinz. Space-efficient indexing of Chess endgame tables. *Journal of the International Computer Games Association*, 23(3):148–162, 2000.

[Newell *et al.*, 1958] Allen Newell, J. C. Shaw, and Herbert A. Simon. Chess-playing programs and the problem of complexity. *IBM Journal of Research and Development*, 2(4):320–335, 1958.

[Patashnik, 1980] Oren Patashnik. Qubic: 4x4x4 tic-tac-toe. *Mathematics Magazine*, 53(4):202–216, 1980.

[Quinlan, 1983] John Ross Quinlan. Learning efficient classification procedures and their application to Chess end games. In *Machine Learning: an Artificial Intelligence Approach*, pages 463–482, 1983.

[Sarkar, 1996] Debranjan Sarkar. Boolean function-based approach for encoding of binary images. *Pattern Recognition Letters*, 17(8):839 – 848, 1996.

[Schaeffer *et al.*, 2007] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007.

[Shannon, 1950] Claude E. Shannon. A Chess-playing machine. *Scientific American*, 182(2):48–51, February 1950.

[Silver *et al.*, 2018] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[Stiller, 1989] Lewis Stiller. Parallel analysis of certain endgames. *The Journal of the International Computer Chess Association*, 12(2):55–64, 1989.

[Stiller, 1991] Lewis Stiller. Group graphs and computational symmetry on massively parallel architecture. *The Journal of Supercomputing*, 5(2-3):99–117, 1991.

[Stiller, 1996] Lewis Stiller. Multilinear algebra and Chess endgames. *Games of no chance*, 29:151–192, 1996.

[Ströhlein, 1970] T. Ströhlein. *Untersuchungen über Kombinatorische Spiele ("Investigations on Combinatorial Games")*. PhD thesis, Fakultät für Allgemeine Wissenschaften der Technische Hochschule München ("Faculty of General Sciences of Munich Technical University"), 1970.

[Thompson, 1986] Ken Thompson. Retrograde analysis of certain endgames. *The Journal of the International Computer Chess Association*, 9(3):131–139, 1986.

[Turing, 1953] Alan Mathison Turing. Digital computers applied to games. *Faster than Thought*, 1953.

[v. Neumann, 1928] J. v. Neumann. Zur theorie der gesellschaftsspiele ("On the theory of parlor games"). *Mathematische Annalen ("Mathematical Annals")*, 100:295–320, 1928.

[van den Herik and Herschberg, 1985] H.J. van den Herik and I.S. Herschberg. The construction of an omniscient endgame data base. *The Journal of the International Computer Chess Association*, 8(2):66–87, 1985.

[Yang *et al.*, 2006] Jeehong Yang, Serap A. Savari, and Oskar Mencercv. Lossless compression using two-level and multi-level boolean minimization. In *2006 IEEE Workshop on Signal Processing Systems Design and Implementation*, pages 148–152, 2006.

[Zakharov and Maknhychev, 2013] V. Zakharov and V. Maknhychev. Creating tables of Chess 7-piece endgames on the Lomonosov supercomputer. *Superkomp'yutery ("Supercomputers")*, 15, 2013.

[Zermelo, 1913] Ernst Zermelo. 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.