Preprint of an article in: ICCA Journal 22(1), pp. 22–32, Mar. 1999.

Endgame Databases and Efficient Index Schemes

Ernst A. Heinz*
Institute for Program Structures and Data Organization (IPD)
School of Computer Science, University of Karlsruhe
P.O. Box 6980, D-76128 Karlsruhe, F.R. Germany

February 7, 1999

Abstract

Endgame databases have become an integral part of modern chess programs during the past few years. Since the early 1990s two different kinds of endgame databases are publicly available, namely Edwards' so-called "tablebases" and Thompson's collection of 5-piece databases. Although Thompson's databases enjoy much wider international fame, most current chess programs use tablebases because they integrate far better with the rest of the search. For the benefit of all those enthusiasts who intend to incorporate endgame databases into their own chess programs, this article describes the index schemes of Edwards' tablebases and Thompson's databases in detail, explains their differences, and provides a comparative evaluation of both.

In addition we introduce new indexing methods that improve on the published state-of-the-art and shrink the respective index ranges even further (especially for large databases and endgames with Pawns). This reduces the overall space requirements of the resulting databases substantially. We also propose several solutions for the problem of potential en-passant captures in endgame databases with at least one Pawn per each side. Shortly after the initial submission of our original text, Nalimov independently applied similar techniques as ours and even more advanced index schemes to his new tablebases which are now publicly available on the Internet, too. We briefly put his work into perspective as well.

*Email: <heinze@ira.uka.de>, WWW = <http://wwwipd.ira.uka.de/~heinze/>

1 Introduction

As soon as CD-ROM technology became affordable for the mainstream, Thompson made his entire set of endgame databases publicly available on 4 CDs [15, 16, 18] starting in 1991 (see Appendix A for more details about the contents of these famous volumes). In order to squeeze the most out of each CD, he compressed the raw distance-to-conversion data by means of Huffman-encoding per endgame. His final database files therefore contain the compressed distance-to-conversion data preceded by the generated Huffman-tables as needed for decompression. Beside the database files, Thompson's CDs include a test program written in ANSI-C that demonstrates how to access the Huffman-encoded databases. Like a few others, we employed Thompson's compressed databases to enable perfect endgame play by our chess program DarkThought[1] [5] when it participated in the 8th World Computer Chess Championship (WCCC) in Hong Kong, May 1995, and the 13th World Microcomputer Chess Championship (WMCC) in Paderborn, October 1995.

These endeavours quickly revealed the greatest disadvantages of Thompson's endgame databases with respect to their efficient and seamless integration into chess-playing programs. First and foremost, the Huffman-encoding causes unpredictable slow-downs of the database probes because several random accesses to the external database files may be necessary before the desired values get finally retrieved. This happens regardless of whether you fully pre-load the Huffman-tables of all database files into RAM or not. The external file accesses make it practically impossible to probe Thompson's databases at deep levels of the game tree without slowing the search down to a crawl. Moreover, Thompson's databases only host positions where the materially weaker and thence potentially losing side enjoys the right to move. His databases contain exact results for these positions if the weaker side actually loses. Yet draws and possible wins of the weaker side remain unresolved because the databases do not discriminate between them. Consequently, far more than half of the endgame positions that occur during searches and are suitable for database recognition at all do not apply to Thompson's files on average because either the materially stronger side is to move or it does not win. In theory this gets accounted for by 1-ply searches starting at the positions with the wrong side to move. But of course nobody wants to do so in practice, except for maybe at the root node of the search tree. Last but not least, Thompson's distance-to-conversion metric makes

[1] WWW = <http://wwwipd.ira.uka.de/Tichy/DarkThought/>

the values returned by database probes quite hard to use because they must be converted to reasonable scores that do not conflict with real mates.

The sobering disadvantages of Thompson's databases in the practical realm of chess programs made Edwards take a road different from Thompson's. He started to explore endgame databases with the specific intention to integrate the resulting creations into his chess program SPECTOR. Right from the beginning of his work in the early 1990s, Edwards therefore relied on the distance-to-mate metric and the complete coverage of all positions with both sides to move. Furthermore, he designed his so-called "tablebases" as data vectors that allow for the efficient retrieval of database results by means of a single file access. This essential property makes the behaviour of tablebases as predictable as that of other lookup tables. Because the external compression of databases causes unpredictable access behaviour, it was never a viable option for Edwards. He initially constrcuted all 3-piece and 4-piece tablebases as distinct files for positions with either Black or White to move, including KPPK and KPKP without en-passant captures which are both not available from Thompson. After finishing the construction procedure, Edwards made the whole data, a short documentation [3], and a test program written in ANSI-C publicly available on the Internet (see Appendix B for more information about this repository). The tablebases soon became a huge success with many authors of other chess programs. When Hyatt incorporated them into his well-known and widely distributed freeware chess program CRAFTY, they also surged in popularity among the non-programming chess and computer-chess enthusiasts.

Unfortunately, Edwards interest in endgame databases waned over time despite the broad attention and recognition he received. He never completed his work on 5-piece tablebases of which he constructed only a few. But every interested reader may easily follow-up on Edwards' efforts after downloading his publicly available tablebase generator (see Appendix B) in order to construct new 5-piece tablebases all by himself. Before doing so, however, we suggest to continue reading this article first. Beside their many advantages, Edwards' tablebases also feature one severe disadvantage – namely their huge space consumption. In this respect our detailed analyses of index schemes for endgame databases in chess show that Edwards' tablebases are much larger than necessary due to far from optimal indexing. With our proposed improvements their sizes can be reduced substantially by at least 30% in general and by more than half in many special cases. After the completion of our original text, Nalimov applied similar techniques as ours and even more advanced index schemes to his new tablebases (see Appendix C). Meanwhile, the new tablebases and Nalimov's generator program are also publicly available. Because of their higher space efficiency we strongly recommend to use them instead of Edwards' original ones.

2 Related Work

Ströhlein's Ph.D. thesis [14] from 1970 constitutes the earliest known publication that reports about the algorithmic construction of endgame databases in chess. Although Ströhlein's thesis did not immediately gain the broad recognition it deserved, research in the field of endgame databases for chess paced on. Several people independently reinvented the method of *retrograde generation* during the mid-to-late 1970s. This extremely powerful tool quickly reaped fruits and led to the construction of many 3-piece and 4-piece endgame databases as well as some incomplete 5-piece ones (e.g. KPRKR [1]) by different researchers. Due to the limited resources available for computers of this era, it was common practice to break the databases into subsets that were small enough to be handled efficiently. Van den Herik and Herschberg did not only provide a detailed description of retrograde analysis as applied to the endgame KBNK [8] but also gathered an excellent bibliography of related work published up to 1985 inclusively [7].

In 1986 the research efforts culminated in the full construction of all 3-piece and 4-piece as well as many 5-piece endgame databases with at most one Pawn by Thompson [19]. He employed a distance-to-conversion metric and finished the entire set of all not trivially won 5-piece endgame databases with at most one Pawn later on [15, 16, 18]. Our comparisons and discussions in the remainder of this article show, that Thompson's index scheme and assorted index calculations were already very well-tuned. Then Herschberg et al. seeded the work on 6-piece endgame databases in 1987 by describing the first full analysis ever of such a "monster", namely for the subpart KRP(a2)KbBP(a3) of KRPKBP [6]. In 1989 Stiller followed up on Thompson's work and presented a massively parallel implementation of a retrograde generator for endgame databases that reused Thompson's index schemes [13]. In the early 1990s Stiller succeeded in constructing complete databases for selected 6-piece endgames like KQNKRR [11] (see Section 5.2 for further information). Although his generator ran on large parallel supercomputers, he could not save the computed data due to lack of external storage capacity. Only the result summaries and the optimal lines of play for the worst-case winning positions are still available. Thompson's intention [17] to construct all 6-piece databases without Pawns represents the latest development in the distance-to-conversion field.

As for endgame databases relying on the distance-to-mate metric, Edwards constructed all 3-piece and 4-piece tablebases in the early 1990s [2, 3]. But then he suddenly stopped his work on endgame databases in 1996 after completing only a few 5-piece tablebases. Lucky enough, Edwards' efforts were not the only ones in this direction. According to private communication and public announcements on the Newsnet group *rec.games.chess.computer*, Moreland, the author of the chess program FERRET, implemented his own retrograde generator for endgame databases in the mid-1990s. Today, Moreland seems to have constructed all meaningful 3-piece, 4-piece, and 5-piece endgame databases with his own generator while using the distance-to-mate metric. His specific database format resembles that of Edwards' tablebases with somewhat improved indexing for endgames with Pawns. Moreland was so kind as to provide his database generator to us on request. Unfortunately, however, he has not yet made his whole data publicly available.

Shortly after submitting the original version of this text to the *ICCA Journal*, Nalimov announced his active work aimed at the construction of distance-to-mate endgame databases. He made his new tablebases and the source code of his ANSI-C++ retrograde generator publicly available in late 1998. Nalimov's primary concern for the new tablebases was their space-efficient storage. Therefore, he applied even more advanced index schemes than ours in order to reduce the number of elements per tablebase. Nalimov's generator program constructs all 3-piece and 4-piece tablebases in roughly one hour on a fast personal computer while handling potential en-passant captures correctly. As of February 1999, he successfully completed the construction of all 5-piece tablebases featuring King plus two men on one side (including 2x Pawns) versus King plus one man (including a Pawn) on the other. Please see Appendix C for more details about Nalimov's tablebases.

3 Indexing Endgame Databases without Pawns

The most obvious index scheme for endgame databases simply allocates the whole 64 squares of the chess board as the index range for each piece. The number of elements contained in a k-piece endgame database as indexed by this naive scheme equals exactly 64^k. Because the locations of all pieces on the chess board are basically independent from each other, it is provably impossible to make the sizes of endgame databases grow less than exponential in the number of pieces. But we can try to reduce the base of the exponential and the rate of its growth. In this respect, the particular geometry of the chess board allows for the exploitation of several symmetries and other clever tricks. As shown in the sequel, accordingly sophisticated index schemes achieve substantial reductions of the overall number of elements contained in endgame databases.

All moves of pieces other than Pawns are totally reversible as long as they do not change the current castling rights. Because castling is normally no longer possible in endgames anyway, all databases constructed up to now explicitly neglect castling. They are only valid for positions without any castling rights whatsoever. Based on this precondition, all positions of every endgame database without Pawns feature only reversible moves as legal continuations. Hence, the inherent symmetries of the according endgames may be fully exploited by the index schemes of the databases.

Endgames with purely reversible moves are symmetric along the diagonal, horizontal, and vertical middle axes of the chess board. The ensuing 3-fold symmetry[2] allows for the confinement of one piece to a triangle region of the chess board that comprises just 10 squares. It is spanned by half a diagonal and half an edge anchored in the same corner. This effectively reduces the basic index ranges of endgame databases without Pawns by $(64 - 10)/64 = 84.38\%$ or more than 6-fold overall.

Edwards' Tablebases. Edwards uses a1–d1–d4 as his triangle region. For any particular endgame K..K..X, he always confines the last identified piece X to the triangle a1–d1–d4 (e.g. KQKR $\to$ X = black Rook, KQRK $\to$ X = black King). Although this scheme looks natural at a first glance, it is actually quite tricky to handle because different pieces get confined in specific endgames. Furthermore, on closer examination the scheme turns out to be far inferior than that which Thompson had already published much earlier. All in all, more than one quarter of the data in Edwards' tablebases without Pawns is superfluous.

Thompson's Databases. Thompson employs a8–d8–d5 as his triangle region and always confines the white King to it. Next, he exploits an additional symmetry that arises if the confined white King is located on the long diagonal a8–h1. If so, all placements of the opposing black King are symmetric along this very diagonal. Last but not least, Thompson enumerates the strictly legal placements of both the confined white King and the symmetry-reduced black King on an otherwise empty chess board. As compared with the $10 * 64 = 640$ theoretically possible placements, Thompson's strictly legal symmetry-reduced ones amount to just 462 and thus result in $(640 - 462)/640 = 27.81\%$

[2]Solely one of the two diagonal symmetries combines simultaneously with the rest.

savings overall. This means that the index ranges of Edwards' tablebases without Pawns are $(640-462)/462 = 38.53\%$ larger than those of Thompson's databases (see Table 1).

The necessary lookup table of 640 pre-computed combination values for the two Kings occupies merely 1.3 KBytes of memory. But Thompson does not use this minimal lookup table. Instead, he employs a slightly larger one with 1024 elements and a memory consumption of 2 KBytes. The larger table integrates in a better way with his index calculations which he carefully fine-tuned for speedy execution on computers of the early-to-mid 1980s.

4 Indexing Endgame Databases with Pawns

Because of the irreversibility of Pawn moves, endgames with Pawns contain no symmetries along diagonal or horizontal axes of the chess board. But they feature vertical symmetry along the middle of the chess board when you confine one of the pieces to either the King-side flank (KSF) or the Queen-side flank (QSF). Both Edwards and Thompson exploit this KSF/QSF symmetry by confining one Pawn to the Queen-side flank which effectively halves the basic index ranges of endgame databases with Pawns.

Thompson saves additional space by discarding the squares of the back ranks that no Pawn can ever reach without getting promoted. This reduces the index ranges of endgame databases with Pawns by another $(32-24)/32 = 25\%$ per Pawn. Surprisingly enough, Edwards does not employ this trivial index optimization thus rendering a quarter of the data in his tablebases with Pawns superfluous. Consequently, the index ranges of Edwards' tablebases with Pawns are $(32-24)/24 = 33.33\%$ or exactly by one third larger than those of Thompson's databases (see Table 1).

4.1 The Two Kings

Except for the exploitation of KSF/QSF symmetries and the removal of superfluous back-rank squares, neither Edwards nor Thompson try to further reduce the index ranges of their endgame databases with Pawns. In our own chess program, however, we achieve an additional reduction of 11.82% for the index ranges of endgame databases with Pawns by simply enumerating the strictly legal placements of two Kings on an otherwise empty chess board. As compared with the $64^2 = 4096$ theoretically possible placements of two Kings, all strictly legal ones amount to just 3612 and thus result in $(4096-$

3612)/4096 = 11.82% savings overall (see Table 1). The necessary lookup table *legal_kk_enum* of 4096 pre-computed enumeration values occupies only 8 KBytes of memory which the 11.82% overall savings offset by far. The 2-dimensional lookup table itself is indexed by the locations K_b and K_w of the two Kings: $kk\_index(K_b, K_w) = legal\_kk\_enum[K_b][K_w]$.

4.2 Directly Rammed Pawns

Another interesting scheme to shrink the index ranges of endgame databases with Pawns restricts their contents to such positions where two Pawns of opposite colours are always rammed directly against each other (e.g. black Pawn on d5, white Pawn on d4). In this case it suffices to include the location of only one of the two Pawns in the index function because it also determines the location of the rammed other one. The restriction to rammed Pawns therefore effectively reduces the dimension of the index space by one which is of special interest for future 6-piece endgame databases. Moreover, rammed Pawns can never reach the rank directly in front of the opponent's back rank (otherwise there would be no ram). This allows us to discard another rank from the index of each pair of rammed Pawns. Overall, two rammed Pawns need $(24 - 20)/24 = 16.67\%$ less space than a single Pawn belonging to one side only. Example: our 3-piece endgame database KPK contains $3612 * 24 = 86688$ positions whereas our 4-piece endgame database $\mathrm{KP_{\triangleright}KP_{\triangleright}}$ with directly rammed Pawns $\mathrm{P_{\triangleright}}$ comprises merely $3612 * 20 = 72240$ (see Table 1).

4.3 En-Passant Captures

Edwards' KPKP tablebase does not account for possible en-passant captures and Thompson did not generate any databases with two Pawns at all. This gap in the set of Edwards' and Thompson's endgame databases seems to have been caused by the additional complexity that pending en-passant captures introduce with respect to indexing and the retrograde process of database construction. Edwards and Thompson probably did not deem the benefits of correct en-passant handling worth the effort to implement the necessary special-case code. To the best of our knowledge, Forthoffer et al. [4] and Herschberg et al. [9] were the first to mention the existence of a retrograde generator by Dekker in 1989 which was presumably able to handle castling moves and en-passant captures correctly. Unfortunately, however, we are not aware of any other publication about Dekker's then exceptional program RETRO. Meanwhile, Moreland and Nalimov successfully constructed several endgame databases with correct handling of en-passant captures after integrating en-passant support into their generators in 1998.

In order to distinguish between "plain" Pawns and those with the pending possibility of en-passant captures, we can simply add the back-rank squares of each side to the index ranges of all Pawns. This addition suffices because en-passant captures are only possible on a single rank for each side (the 5th rank for black Pawns and the 4th rank for white Pawns respectively). Then, Pawns with the pending possibility of en-passant captures get indexed by the back-rank squares of their according file and side. Example: black Pawns on d5 with pending en-passant captures get indexed by d8 and equivalent white Pawns on d4 get indexed by d1. The proposed scheme solves the index problem of pending en-passant captures while increasing the index ranges of endgame databases with Pawns on both sides by $(56-48)/48 = (28-24)/24 = 16.67\%$ per Pawn. For the 4-piece endgame database KP_*KP_* with accordingly correct handling of en-passant captures this amounts to an overall increase by $(28*56-24*48)/(24*48) = 36.11\%$ as compared with its plain yet incorrect sibling (see Table 1).

Fortunately, we can still do much better if we really want to. For the sole purpose of explaining the improvement, we temporarily focus on endgame databases with exactly one Pawn per side. Because one of the two Pawns gets confined to the Queen-side flank, we must accomodate exactly 14 different constellations of the two Pawns that feature possible en-passant captures. We solve this accomodation problem by redefining the meaning of otherwise illegal index combinations (e.g. black and white Pawn on d4). For a QSF-confined white Pawn, the 14 potential en-passant constellations and a sample redefinition mapping read as follows (format: white/black Pawn).[3]

d4/e4 → a2/a2	d4/c4 → b2/b2	c4/d4 → c2/c2	c4/b4 → d2/d2
b4/c4 → a3/a3	b4/a4 → b3/b3	a4/b4 → c3/c3	e5/d5 → d3/d3
d5/c5 → a4/a4	c5/d5 → b4/b4	c5/b5 → c4/c4	b5/c5 → d4/d4
b5/a5 → a5/a5	a5/b5 → b5/b5		

While the somewhat tricky mapping of en-passant constellations to final index values certainly complicates the index calculations, it is well worth the added complexity. Based on this scheme, the endgame databases $KP_\bullet..KP_\bullet..$ with correct handling of en-passant captures and at least one Pawn per side consume no more space than their plain yet incorrect siblings (see Table 1). Although we only explained it for endgames with exactly one Pawn per side,

[3]This constitutes an optimized specialization of the enumeration technique.

the scheme actually applies to all endgames with Pawns on both sides. The general applicability springs from the fact that we can always arrange the Pawns in such a way as to place pending en-passant constellations first in the index list. Because only a single en-passant capture can pend at any time, it is well-defined and unique. Moreover, there are Pawns of both sides involved in any en-passant capture such that we can always confine one of them to the Queen-side flank.

5 Further Indexing Improvements for Endgame Databases in General

Researchers have used the straightforward trick to confine certain pieces to subsets of all squares (e.g. Pawn on h7 or only on the h-file) since the early days of endgame databases in computer chess. Thereby one can easily fit most any subparts of all databases to the available computing resources. Nowadays, such decomposotion is of practical interest for 6-piece endgame databases as soon as they become available because the sizes of their complete data still exceed the capacities of all common storage devices by far.

5.1 Equal Locations

Another possibility geared especially towards large endgame databases is to enumerate additional pieces together with the two Kings. For the first such piece this results in an overall reduction of the index space by $(64-62)/64 = 3.13\%$ because the two Kings limit the legal placements of the piece (see Table 1). The first Pawn does not benefit from this because its index range lacks the two back ranks. But in general all the already located Kings, pieces, and Pawns of both sides restrict the legal placements of any additional men.

- For the 2nd Pawn without en-passant handling this results in an overall reduction of the index space by $(48 - 47)/48 = 2.08\%$ (see Table 1).
- For the 2nd piece this results in savings of $(64-61)/64 = 4.69\%$ and a cumulative overall reduction of the index space together with the first piece by $(64 * 64 - 62 * 61)/(64 * 64) = 7.67\%$ (see Table 1).
- For the 3rd Pawn without en-passant handling this results in savings of $(48 - 46)/48 = 4.17\%$. The cumulative overall reduction of the index space together with the other Pawns exceeds the mechanically expected $(24*48*48-24*47*46)/(24*48*48) = 6.16\%$ because here

one side features at least two Pawns which enable additional index optimizations for equal pieces as explained in Section 5.2.

- For the 3rd piece this results in savings of $(64 - 60)/64 = 6.25\%$ and a cumulative overall reduction of the index space together with the other pieces by $(64 * 64 * 64 - 62 * 61 * 60)/(64 * 64 * 64) = 13.44\%$.
- For the 4th Pawn without en-passant handling this results in savings of $(48 - 45)/48 = 6.25\%$. The cumulative overall reduction of the index space together with the other Pawns exceeds the mechanically expected $(24*48*48*48-24*47*46*45)/(24*48*48*48) = 12.03\%$ because here one side features at least two Pawns which enable additional index optimizations for equal pieces as explained in Section 5.2.
- For the 4th piece this results in savings of $(64-59)/64 = 7.81\%$ and a cumulative overall reduction of the index space together with the other pieces by $(64*64*64*64-62*61*60*59)/(64*64*64*64) = 20.20\%$.

Implementing the above enumerations with the help of lookup tables turns out to be quite naive and wasteful because the necessary tables get very large. The first piece of endgame databases with Pawns, for example, requires an enumeration table with $3612 * 64 = 231168$ entries that occupy 903 KBytes of memory (32-bit values). For every additional piece the lookup tables grow exponentially in size and quickly erase the gains of the enumeration which then does not pay off overall. Fortunately, Nalimov made us aware of how to perform the desired enumerations for each Pawn or piece “on the fly” during their respective index calculations.

- The index of the n-th Pawn P_n equals $I(P_n) = 48 - \sum_{j=1}^{n-1}(P_n > P_j)$.
- The index $I(X_n)$ of the n-th piece X_n depends on the two Kings K_b, K_w of Black and White as well as on the the m Pawns of both sides taken together such that it translates to $I(X_n) = 64 - (X_n > K_b) - (X_n > K_w) - \sum_{j=1}^{m}(X_n > P_j) - \sum_{j=1}^{n-1}(X_n > X_j)$.

The convincing advantages of this dynamic enumeration scheme spring from its great efficiency. The dynamic enumeration does not consume any additional memory at all while being easy to implement and fast to compute.

En-Passant Captures Revisited

The tightening of the index space by removing illegal placements of pieces and Pawns (see above) invalidates our en-passant solutions from Section 4.3.

The easiest way to accomodate the 14 en-passant constellations while still keeping the index space tight is to add them to the enumeration range of the first two Pawns of opposite colours.[4] This results in 7 additional enumeration values or $(24 * 47) + 7 = 1135$ altogether because each Black and White are to move in exactly half of the 14 en-passant constellations. Based thereon, the endgame databases $\mathrm{KP}_\diamond..\mathrm{KP}_\diamond..$ with tight index spaces, correct handling of en-passant captures, and at least one Pawn per side consume only $7/(24 * 47) = 0.6\%$ more space than their plain yet incorrect siblings (see Table 1).

5.2 Equal Pieces

Thompson briefly refers to "potentially other symmetries" in his article about 5-piece endgame databases [19] but he neither exploits them for indexing nor does he explain them in detail. He simply mentions that "if there are two white Rooks, they could exchange places without altering the position". In the article about his endeavour to generate 6-piece endgame databases [17], he further speculates that Stiller "ran the KR-BKNN endgame [...] making use of the Knight-Knight symmetry, thereby reducing the required memory to half". Although Stiller never explicitly mentions this improvement for double pieces per side, his general group-theoretic framework for the exploitation of symmetry as developed during his Ph.D. studies [10, 12] clearly subsumes this optimization. In contrast to both Edwards' and Thompson's database generators, Stiller's program therefore surely employed it.[5]

In sole terms of chess this index optimization means that the sequences in which you place two equal pieces on a chess board do not matter at all as for the resulting positions – they are always identical. Example: two Rooks of the same colour on d1 and e1 may be placed in either order (i.e. d1 after e1 or e1 after d1) without affecting the final placement of both. In practice we diagonalize the index space of two equal pieces by reordering the pair in such a way that the piece with the larger location value always comes first. This results in $\sum_{j=1}^{63}(j) = 2016$ distinct placements of two equal pieces on an otherwise empty chess board, thus saving $(4096 - 2016)/4096 = 50.78\%$ per pair as compared with the $64^2 = 4096$ theoretically possible ones. The index ranges of equal pieces shrink even further if you combine the diagonalization

[4] In a similar way, we could account for any potential castling rights by adding them to the combined enumeration ranges of the according Kings and Rooks.

[5] We strongly suspect that Zellner's KBBK database generator [20] running on a Commodore C64 computer with 64 KBytes of RAM applied the optimization, too.

with the index reduction for illegal placements of pieces on equal squares (see Section 5.1). By taking care of the omnipresent Kings, for instance, the combined index optimizations result in merely $\sum_{j=1}^{61}(j) = 1891$ distinct placements of two equal pieces which saves $(4096 - 1891)/4096 = 53.83\%$ space per pair in comparison with the normal indexing method (see Table 1).

Pawns allow for the same kind of optimization with the exception of the first pair in the endgame classes KPP..K[PP].. and K[PP]..KPP.. with two or more Pawns per side if any at all (see next subsection). For other pairs of Pawns without correct en-passant handling, the diagonalization results in at most $\sum_{j=1}^{46}(j) = 1081$ distinct placements of both Pawns. This shrinks the combined index range of the two Pawns by at least $(2304 - 1081)/2304 = 53.08\%$ as compared with the $48 * 48 = 2304$ theoretically possible constellations on an otherwise empty chess board.

With respect to the index reduction for illegal placements of pieces on equal squares alone, the diagonalization generally saves exactly 50% of space as easily proven hereafter. Let $m > 1$ denote the number of possible placements for the first of the two equal pieces. Then, the tight yet non-diagonalized index range of the two pieces equals $m * (m - 1)$. Their tight and diagonalized index range, however, amounts to $\sum_{j=1}^{m-1}(j) = \frac{1}{2} * (m - 1) * (m - 1 + 1)$ which is exactly half of the non-diagonalized one.

The application of the combined index optimization is particularly interesting for the KXXKYY class of 6-piece endgame databases with two pairs of equal pieces (X, Y = B, N, R, Q). Here, the index range of the second pair shrinks to $\sum_{j=1}^{59}(j) = 1770$ which is $(4096 - 1770)/4096 = 56.79\%$ less than normal. Both pairs of equal pieces together reduce the space consumption of KXXKYY endgame databases by sizable $(4096 * 4096 - 1891 * 1770)/(4096 * 4096) = 80.05\%$ overall as compared with the standard indexing scheme. In the same way as for the Kings, we enumerate the placements of two equal pieces into 2-dimensional lookup tables with maximally 4096 elements each. The tables occupy at most 8 KBytes of memory and can be used for all pairs. The according tables for Pawns comprise maximally 2304 elements and need at most 4.5 KBytes of memory.

The Peculiarities of KPP. . K[PP]. . and K[PP]. . KPP. .

For the special cases of KPP..K[PP].. and K[PP]..KPP.. with two or more Pawns per side if any at all, the full power of the diagonalizing optimization for equal pieces remains untapped as long as we confine one of the Pawns to the Queen-side flank (QSF). Because the index range of this QSF-Pawn gets halved, we can only do a pseudo-diagonalization for it that is close to but not really optimal. For KPPK this results in a combined index range of $\sum_{r=0}^{5}\sum_{f=0}^{3}(8*r+2*f+1) = 576$ for the two Pawns where the optimum equals $(24*47)/2 = 564$. Nonetheless, the pseudo-diagonalization still fares amazing $(32*64*64*64-3612*576)/(32*64*64*64) = 75.20\%$ better overall than Edwards' index scheme (see Table 1). The 2-dimensional lookup table for the necessary enumeration spans $24*48 = 1152$ elements and requires 2.25 KBytes of memory.

5.3 Equal Material

Our final optimization relates to endgame databases such as Edwards' tablebases that contain the complete results for both sides enjoying the right to move in each position. These databases normally consist of two distinct subparts, namely one for all positions with Black-to-move (BTM) and another one for those with White-to-move (WTM). Now, if both sides feature equal material constellations like KPKP, KQKQ, KQPKQP etc., one of the two subparts is clearly superfluous. In this case, any position with BTM can easily be mapped to an equivalent position with WTM and vice versa by mirroring all pieces at the horizontal middle axis of the chess board (e.g. d2 gets mirrored to $\mathrm{d7} = \mathrm{d2}^{mirr}$ and b6 gets mirrored to $\mathrm{b3} = \mathrm{b6}^{mirr}$). We let $\mathrm{KX}_1 \cdots \mathrm{X}_n\mathrm{KY}_1 \cdots \mathrm{Y}_n$ (where $\mathrm{X}_i = \mathrm{Y}_i$ for all $i \in \{1 \ldots n\}$) denote the class of endgame databases with equal material for both sides. Then, the following equation formalizes our verbal statement about the equivalence of database queries *edb_val*() for mirrored positions ($\mathrm{K}_b, \mathrm{K}_w$ = Kings of Black and White respectively).

$$\boxed{\begin{gathered} \mathit{edb\_val}(\mathrm{BTM},\ K_w,\ X_1,\ \ldots,\ X_n,\ K_b,\ Y_1,\ \ldots,\ Y_n) \\ = \\ \mathit{edb\_val}(\mathrm{WTM},\ K_b^{mirr},\ Y_1^{mirr},\ \ldots,\ Y_n^{mirr},\ K_w^{mirr},\ X_1^{mirr},\ \ldots,\ X_n^{mirr}) \end{gathered}}$$

Consequently, one single subpart of positions with only BTM or WTM suffices to characterize the whole endgame database if both sides have equal material. Unfortunately, Edwards does not exploit this fact and always stores both parts of his tablebases although only one is really needed.

6 Discussion and Conclusion

In contrast to Thompson's endgame databases, Edwards' tablebases are well-suited for integration into a chess program that probes them everywhere in the search tree. But due to Edwards' suboptimal choices of index schemes, his tablebases contain a lot of superfluous data and thus waste much precious space. Table 1 summarizes the specific differences between Edwards' and Thompson's index schemes for all 3-piece and 4-piece endgames where X and Y denote any pieces other than Kings and Pawns. The table uses Thompson's indexing as the primary reference of the comparisons. It also includes our new index schemes as introduced throughout the article. The comparison clearly shows the spaciousness of Edwards' tablebases and the strict dominance of our new index schemes over the two others.

Endgame Database	**Edwards #Elements**	**Edwards Δ Elem.**	**Thompson #Elements**	**Heinz Δ Elem.**	**Heinz #Elements**
KPK	$32 * 64 * 64$	+33.33%	$24 * 64 * 64$	−11.82%	$3612 * 24$
KXK	$10 * 64 * 64$	+38.53%	$462 * 64$	−3.13%	$462 * 62$
KP$_\triangleright$KP$_\triangleright$	n.a.	− − −	n.a.	− − −	$3612 * 20$
KP$_*$KP$_*$	n.a.	− − −	n.a.	− − −	$3612 * 28 * 56$
KP$_\bullet$KP$_\bullet$	n.a.	− − −	n.a.	− − −	$3612 * 24 * 48$
KP$_\diamond$KP$_\diamond$	n.a.	− − −	n.a.	− − −	$3612 * 1135$
KPKP	$32 * 64 * 64 * 64$	− − −	n.a.	−51.43%	$3612 * 24 * 47$
KPPK	$32 * 64 * 64 * 64$	− − −	n.a.	−75.20%	$3612 * 576$
KPKX	$32 * 64 * 64 * 64$	+33.33%	$24 * 64 * 64 * 64$	−15.95%	$3612 * 24 * 61$
KPXK	$32 * 64 * 64 * 64$	+33.33%	$24 * 64 * 64 * 64$	−15.95%	$3612 * 24 * 61$
KXXK	$10 * 64 * 64 * 64$	+38.53%	$462 * 64 * 64$	−53.83%	$462 * 1891$
KXYK	$10 * 64 * 64 * 64$	+38.53%	$462 * 64 * 64$	−7.67%	$462 * 62 * 61$
KXKY	$10 * 64 * 64 * 64$	+38.53%	$462 * 64 * 64$	−7.67%	$462 * 62 * 61$

Table 1: Comparison of Index Schemes for 3-Piece and 4-Piece Databases.

Most of the new index schemes proved their practical viability under tough tournament conditions during the 15th World Microcomputer Chess Championship in October 1997 where our chess program DarkThought employed them. Although our new index functions require slightly more computational effort than the other ones (additional levels of indirection as introduced by the enumeration tables and general multiplications instead of shifts for multiplying with powers of 2), we never observed any notable slow-down while applying them as opposed to not doing so. Of course, we also needed modified endgame databases built around the new index schemes in order to apply them. We derived the modified databases automatically from Edwards' tablebases with a double-checking conversion program that we implemented specifically for this purpose.

Last but not least, we like to stress that shrinking the index spaces of endgame databases is indeed a meaningful contribution to the state-of-the-art. Even in today's technological environment of ever increasing storage capacities at continuously lower prices, memory is still precious for various applications. In the realm of endgame databases such applications are the construction and permanent recording of huge 6-piece databases as well as stand-alone chess computers with no hard disks and only little RAM/ROM or special-purpose chess processors like the ones powering IBM's chess machine DEEP BLUE. These applications greatly benefit from all possible savings of space consumption. For database construction, smaller index spaces offer the additional advantage of possibly reducing the required amount of computation because there may simply be fewer elements to process.

A Thompson's Endgame Databases on CD-ROM

Until he finally ran out of supply in 1996, Thompson himself distributed his endgame databases on CDs to anybody who expressed interest in them. Right now, the cheapest and easiest way to aquire Thompson's complete set of endgame databases seems to buy Meyer-Kahlen's commercial chess program MEPHISTO SHREDDER that includes Thompson's 4 CDs and sells for 99 DM $\approx$ 55 US-$ in Germany. The whole SHREDDER family and their commercial competitors FRITZ 4 & FRITZ 5 by Morsch belong to the small group of strong chess programs that still use Thompson's databases. Of course, they probe the compressed databases only at the root position.

Below follows a complete list and a cross-reference of all 5-piece endgame databases as constructed by Thompson and available on each CD. The given database identifiers name the pieces of the potentially winning side followed by an "_" character and the pieces of the opponent while omitting both Kings (e.g. BB_N denotes KBBKN). The numbers in brackets of the cross-reference tell you which CDs contain the according database. Some are actually present multiple times in order to make each CD self-contained. Please also note that Thompson's 5-piece databases include all of their 3-piece and 4-piece sub-games except for those resulting from promotions of Pawns. By assigning the location of the potentially winning King to all pieces that are no longer present on the board, you can easily probe all sub-games of any database in a convenient way without switching files.

Thompson CD #1: BB_N, BN_N, NN_N, NP_N, Q_BB, Q_BN, QB_Q, QN_N, Q_NN, QN_Q, QP_Q, QQ_Q, Q_RB, Q_RN, QR_Q, QR_R, Q_RR, RB_R, RN_N, RN_R, RP_R, RR_R.

Thompson CD #2: BB_B, BB_N, BN_B, BN_N, BP_B, BP_N, NN_B, NNN_, NP_B, QB_B, QB_N, QN_B, QR_Q, RB_B, RB_N, RB_Q, RN_B, RN_Q, RP_Q, RR_Q.

Thompson CD #3: NN_B, NN_N, NN_P, NN_Q, NN_R, Q_QR, QR_B, Q_RB, Q_RN, Q_RP, Q_RR, RB_B, R_BB, R_BN, R_BP, RN_B, R_NN, R_NP, RP_B, R_QB, R_QN, RR_B, R_RB, R_RN.

Thompson CD #4: BN_B, BN_N, BN_P, BN_Q, BN_R, Q_BB, Q_BN, Q_BP, Q_QB, Q_RB, QR_N, RB_N, RN_N, RP_N, RR_N.

Cross-Reference: BB_B[2], BB_N[1:2], BN_B[2:4], BN_N[1:2:4], BN_P[4], BN_Q[4], BN_R[4], BP_B[2], BP_N[2], NN_B[2:3], NNN_[2], NN_N[1:3], NN_P[3], NN_Q[3], NN_R[3], NP_B[2], NP_N[1], QB_B[2], Q_BB[1:4], QB_N[2], Q_BN[1:4], Q_BP[4], QB_Q[1], QN_B[2], QN_N[1], Q_NN[1], QN_Q[1], QP_Q[1], Q_QB[4], QQ_Q[1], Q_QR[3], QR_B[3], Q_RB[1:3:4], QR_N[4], Q_RN[1:3:4], Q_RP[3], QR_Q[1:2], QR_R[1], Q_RR[1:3], RB_B[2:3], R_BB[3], RB_N[2:4], R_BN[3], R_BP[3], RB_Q[2], RB_R[1], RN_B[2:3], RN_N[1:4], R_NN[3], R_NP[3], RN_Q[2], RN_R[1], RP_B[3], RP_N[4], RP_Q[2], RP_R[1], R_QB[3], R_QN[3], RR_B[3], R_RB[3], R_RN[3], RR_N[4], RR_Q[2], RR_R[1].

B Edwards' Tablebases on the Internet

Although Edwards' tablebases and the source code of his ANSI-C retrograde generator traditionally resided with the Internet Chess Library at URL <ftp://chess.onenet.net/pub/chess/TB/>, they were available from other sites as well. Prior to the advent of Nalimov's tablebases, many strong chess programs employed Edwards' tablebases. Among others this included Hirsch's commercial MChess 7 and Hyatt's popular Crafty freeware up to version 15.20 inclusively. Below follows a complete list of all tablebases as constructed by Edwards.

3-Piece Tablebases: B_, N_, Q_, P_, R_.

4-Piece Tablebases: BB_, B_B, BN_, B_N, BP_, B_P, NN_, N_N, NP_, N_P, PP_, P_P, Q_B, QB_, QN_, Q_N, QP_, Q_P, QQ_, Q_Q, QR_, Q_R, RB_, R_B, RN_, R_N, RP_, R_P, RR_, R_R.

5-Piece Tablebases: BBB_, BB_B, BB_N, BN_B, BN_N, NN_B, NNN_, NN_N, NN_Q, NN_R, QB_Q, QN_Q, QQQ_, QQ_Q, QR_Q, QR_R, RB_B, RB_R, RN_R, RP_R, RRR_, RR_R.

C Nalimov's Tablebases on the Internet

Nalimov's tablebases and the ANSI-C++ source code of his retrograde generator traditionally reside on the FTP server of Crafty at the URL <ftp://ftp.cis.uab.edu/pub/hyatt/TB/>. Because of their high space efficiency, these new tablebases look like they will become the future standard. Whittington's commercial chess program Chess-System Tal 2 and Hyatt's popular Crafty freeware starting with version 16.0 already use them. Nalimov's following new tablebases were available as of February 6, 1999.

3-Piece Tablebases: B_, N_, Q_, P_, R_.

4-Piece Tablebases: BB_, B_B, BN_, B_N, BP_, B_P, NN_, N_N, NP_, N_P, PP_, P_P, Q_B, QB_, QN_, Q_N, QP_, Q_P, QQ_, Q_Q, QR_, Q_R, RB_, R_B, RN_, R_N, RP_, R_P, RR_, R_R.

5-Piece Tablebases: BB_B, BB_N, BB_P, BB_Q, BB_R, BN_B, BN_N, BN_P, BN_Q, BN_R, BP_B, BP_N, BP_P, BP_Q, BP_R, NN_B, NN_N, NN_P, NN_Q, NN_R, NP_B, NP_N, NP_P, NP_Q, NP_R, PP_B, PP_N, PP_P, PP_Q, PP_R, QB_B, QB_N, QB_P, QB_Q, QB_R, QN_B, QN_N, QN_P, QN_Q, QN_R, QP_B, QP_N, QP_P, QP_Q, QP_R, QQ_B, QQ_N, QQ_P, QQ_Q, QQ_R, QR_B, QR_N, QR_P, QR_Q, QR_R, RB_B, RB_N, RB_P, RB_Q, RB_R, RN_B, RN_N, RN_P, RN_Q, RN_R, RP_B, RP_N, RP_P, RP_Q, RP_R, RR_B, RR_N, RR_P, RR_Q, RR_R

References

[1] Arlazarov, V. L. and Futer, A. L. (1979). Computer analysis of a Rook end-game. *Machine Intelligence 9*, J. E. Hayes and D. Michie and L. I. Mikulich (eds.), pp. 361–371, Ellis Horwood, ISBN 0-470-26714-3/0-853-12112-5 [reprinted in *Computer Chess Compendium*, D. N. L. Levy (ed.), pp. 330–336, Springer, 1989, ISBN 0-387-91331-9].

[2] Edwards, S. J. (1995). Comments on Barth's article "Combining knowledge and search to yield infallible endgame programs". *ICCA Journal*, Vol. 18, No. 4, pp. 219–225.

[3] Edwards, S. J. (1994). *Tablebase Documentation*. Electronic document from URL <ftp://chess.onenet.net/pub/chess/TB/README-TB>.

[4] Forthoffer, D. and Rasmussen, L. and Dekker, S. (1989). A correction to some KRKB-database results. *ICCA Journal*, Vol. 12, No. 1, pp. 25–27.

[5] Heinz, E. A. (1997). How DarkThought plays chess. *ICCA Journal*, Vol. 20, No. 3, pp. 166–176.

[6] van den Herik, H. J. and Herschberg, I. S. and Nakad, N. (1987). A six-men-endgame database: KRP(a2)KbBP(a3). *ICCA Journal*, Vol. 10, No. 4, pp. 163–180.

[7] van den Herik, H. J. and Herschberg, I. S. (1986). A data base on data bases. *ICCA Journal*, Vol. 9, No. 1, pp. 29–34.

[8] van den Herik, H. J. and Herschberg, I. S. (1985). The construction of an omniscient endgame data base. *ICCA Journal*, Vol. 8, No. 2, pp. 66–87.

[9] Herschberg, I. S. and van den Herik, H. J. and Schoo, P. N. A. (1989). Verifying and codifying strategies in the KNNKP(h) endgame. *ICCA Journal*, Vol. 12, No. 3, pp. 144–154 [reprinted in *Computers, Chess, and Cognition*, T. A. Marsland and J. Schaeffer (eds.), pp. 183–196, Springer, 1990, ISBN 0-387-97415-6/3-540-97415-6].

[10] Stiller, L. B. (1995). *Exploiting Symmetry of Parallel Architectures*. Ph.D. Thesis, John Hopkins University.

[11] Stiller, L. B. (1994). Multilinear algebra and chess endgames. *Games of No Chance*, R. J. Nowakowski (ed.), MSRI Publications Vol. 29, Cambridge University Press, ISBN 0-521-57411-0.

[12] Stiller, L. B. (1991). Group graphs and computational symmetry on massively parallel architectures. *The Journal of Supercomputing*, Vol. 5, No. 2, pp. 99–117.

[13] Stiller, L. B. (1989). Parallel analysis of certain endgames. *ICCA Journal*, Vol. 12, No. 2, pp. 55–64.

[14] Ströhlein, T. (1970). *Untersuchungen über kombinatorische Spiele*. Dissertation (Ph.D. Thesis), Technical University of Munich.

[15] The Editors. (1993). Thompson: Quintets with variations. *ICCA Journal*, Vol. 16, No. 2, pp. 86–90.

[16] The Editors. (1992). Thompson: All about five men. *ICCA Journal*, Vol. 15, No. 3, pp. 140–143.

[17] Thompson, K. (1996). 6-Piece endgames. *ICCA Journal*, Vol. 19, No. 4, pp. 215–226.

[18] Thompson, K. (1991). Chess endgames volume 1. *ICCA Journal*, Vol. 14, No. 1, p. 22.

[19] Thompson, K. (1986). Retrograde analysis of certain endgames. *ICCA Journal*, Vol. 9, No. 3, pp. 131–139.

[20] Zellner, H. (1986). Compressing databases down to micro size. *ICCA Journal*, Vol. 9, No. 4, pp. 199–200.