The maximum number of legal moves of Othello in reachable positions is 33

Hiroki Takizawa[1]

[1]**Preferred Networks, Inc., Chiyoda-ku, Tokyo, Japan**

Corresponding author:
Hiroki Takizawa[1]

Email address: contact@hiroki-takizawa.name

ABSTRACT

We prove that the maximum number of moves among all reachable positions in a game of Othello is 33. We also construct a game record to achieve such a position. Additionally, including positions that are not reachable from the initial position, we prove that this maximum is 34 and construct such a position. We utilized Satisfiability (SAT) and Satisfiability Modulo Theories (SMT) solvers to obtain these computational proofs. These new findings not only clarify properties of the game of Othello, but also provide hints for the efficient and robust implementation of Othello software, and also present practical examples of using SAT and SMT solvers.

Keywords: Othello, Branching factor, SAT Solver, SMT Solver

INTRODUCTION

Othello is a well-known game, and the development of Othello software has continued for many years. One of the earliest superhuman-strength programs, Logistello, trained through self-play and overpowered the human world champion, Takeshi Murakami, in 1997 (Buro, 1999). Even after Othello software surpassed human performance, efforts to develop even stronger software have persistently progressed, with Edax serving as a prime example (Delorme, 2021).

In order to enhance Othello software, it is essential to optimize search efficiency and achieve highly efficient implementation. In alpha-beta search (Abramson, 1989), move ordering is pivotal to the efficiency. Therefore, a technique is widely adopted that involves enumerating legal moves, quickly evaluating the potential of each move, and then conducting the search in descending order of each move's potential. If the number of legal moves, often referred to as the out-degree or branching factor in technical terms, is 32 or fewer in any given position, the array to store them can be set to a length of 32. This can potentially speed up the software. However, until now, the maximum number of legal moves in Othello has been unknown.

In this paper, we prove that the maximum number of legal moves in all position of Othello that are reachable from the initial position is 33. We also prove that this number increases to 34 if we do not restrict ourselves to positions that are reachable from the initial position. We utilize Satisfiability (SAT) and Satisfiability Modulo Theories (SMT) solver to generate these proofs. Additionally, we reconstruct a game record from our proof, which transitions from the initial position to one with 33 legal moves shown in Figure 1. To the best of our knowledge, these discoveries and computational proofs are novel contributions presented in this study.

These findings not only offer the first clarification of one aspect of the nature of the game of Othello, but also represent successful practical applications of solvers for Satisfiability problems.

RELATED WORKS

The concept of computers playing board games can be traced back at least to Claude Shannon's considerations in 1950 (Shannon, 1950). Even during the 20th century, superhuman-strength software had emerged for many popular board games, including Othello (Buro, 1999), Chess (Campbell et al., 2002),

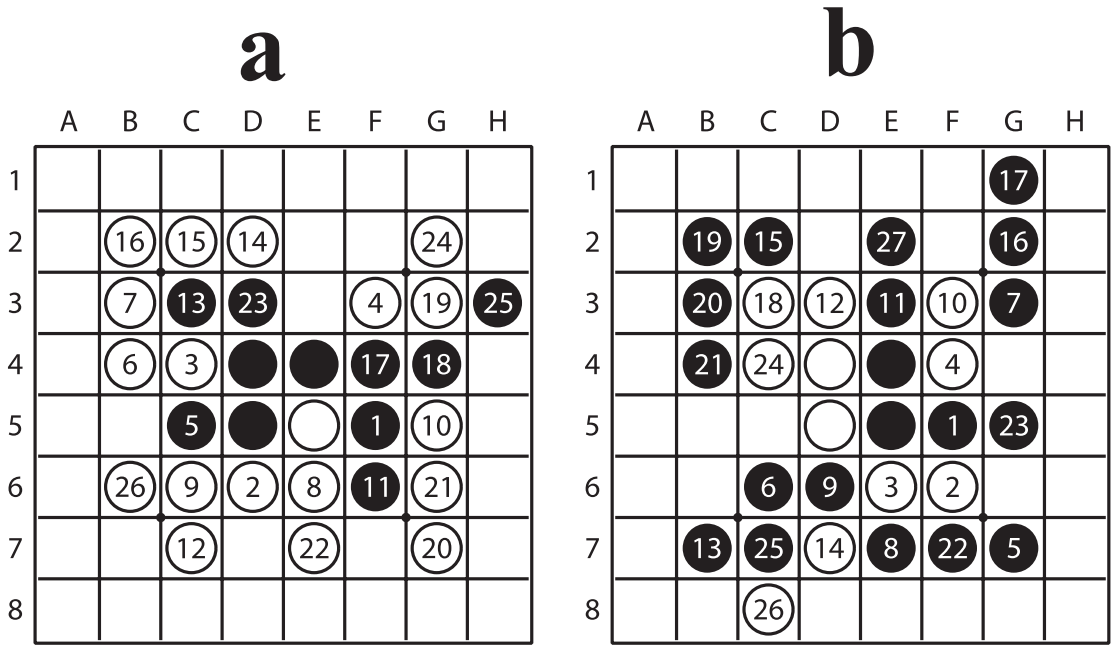

Figure 1. These are positions with 33 legal moves, which were outputted by the SAT solver kissat (Biere and Fleury, 2022), and are reachable from the initial position. (a) A position in which it's Black's turn, there are 33 legal moves, and there are 34 empty squares. The game record for this position is: "F5D6C4F3C5B4B3E6C6G5F6C7C3D2C2B2F4G4G3G7G6E7D3G2H3B6". (b) A position in which it's White's turn and there are 33 legal moves with an equal number of empty squares. The game record for this position is: "F5F6E6F4G7C6G3E7D6F3E3D3B7D7C2G2G1C3B2B3B4F7G5C4C7C8E2".

and Checkers (Schaeffer et al., 1996). These were based on alpha-beta search algorithm (Abramson, 1989), with their strength deriving from the ability to quickly search through a vast number of positions via efficient implementation. In recent years, thanks to improvements in computer performance and advancements of machine learning technology, software has achieved superhuman-strength in a wider range of games, including Go (Silver et al., 2016) and Shogi (Japanese chess) (Kaneko and Takizawa, 2019; Silver et al., 2018).

METHODS

The use of terms ply and move

In chess, there is a convention whereby a pair of consecutive moves, one by white and one by black, is referred to as a "move" or "full-move", while a single move is termed a "ply" or "half-move". However, for clarity in this paper, we do not employ the term "ply" and consistently use the term "move" to denote a single move.

Preliminary

Here, as a preliminary step, we will elucidate the methods for implementing several functions that form the components of the Othello software, in a manner that is conducive to input into an SMT solver.

Bitboard

In this study, an arbitrary Othello position is represented by two 64-bit unsigned integers. Specifically,

- Each bit of the 64-bit integer is bijectively mapped to each square on the Othello board. The least significant bit corresponds to A1, with the A row mapped in sequence to the lowest 8 bits, and the most significant bit corresponds to H8. Generally, when $i = 8j + k$ (where $0 \leq i < 64$ and $0 \leq j,k < 8$), the i-th bit corresponds to the square whose name is the concatenation of the $(j+1)$-th letter of the alphabet with the number $(k+1)$.

- In one of the 64-bit integers, the standing bits correspond to the squares where black stones are located. In the other 64-bit integer, the bits stand in the same manner, but correspond to the squares where white stones are located.

Function to Generate Moves

Assuming that the position is represented by the aforementioned two 64-bit integers, we designate the bitboard of the side whose turn it is as "player", and the bitboard of the opposing side as "opponent". The "get_moves" function shown in Figure 2 returns a bitboard wherein the set bits correspond to all squares where a stone can be legally placed. This "get_moves" function is implemented using only arithmetic operations, bit operations, and logical shifts. This code is available at GitHub: https://github.com/eukaryo/reversi33 .)

```
uint64_t get_some_moves(const uint64_t player, const uint64_t mask, const int dir)
{
    uint64_t flip_l, flip_r;
    uint64_t mask_l, mask_r;
    const int dir2 = dir + dir;

    flip_l = mask & (player << dir);        flip_r = mask & (player >> dir);
    flip_l |= mask & (flip_l << dir);       flip_r |= mask & (flip_r >> dir);
    mask_l = mask & (mask << dir);          mask_r = mask & (mask >> dir);
    flip_l |= mask_l & (flip_l << dir2);    flip_r |= mask_r & (flip_r >> dir2);
    flip_l |= mask_l & (flip_l << dir2);    flip_r |= mask_r & (flip_r >> dir2);

    return (flip_l << dir) | (flip_r >> dir);
}
uint64_t get_moves(const uint64_t player, const uint64_t opponent)
{
    const uint64_t mask = opponent & 0x7E7E7E7E7E7E7E7Eull;

    return (get_some_moves(player, mask, 1)
        | get_some_moves(player, opponent, 8)
        | get_some_moves(player, mask, 7)
        | get_some_moves(player, mask, 9))
        & ~(player | opponent);
}
```

Figure 2. A C++ implementation of a function to generate moves. This is based on Edax 4.4 (Delorme, 2021) with modifications by the author. Edax 4.4 is licenced under GNU General Public License v3.0 (Free Software Foundation, 2007).

Function for Population Count

A function that counts the number of set bits in a 64-bit integer can be implemented using only arithmetic and bit operations, as exemplified by the "bit_count" function in Figure 3. In Figure 3, 64-bit integer multiplication is utilized. This choice is motivated by the fact that some recent CPUs have very low latency for integer multiplication. This calculation has been known for a long time, and an implementation without using integer multiplication is also feasible (Reingold et al., 1977). This code is also available at GitHub: https://github.com/eukaryo/reversi33 .)

Function to Identify the Positions of Stones to Be Flipped

The function that takes the bitboards representing the current player's stones (where "player" denotes the one whose turn it is) and the opponent's stones, along with an integer specifying the move coordinates, and generates the bitboard corresponding to the stones that will be flipped (hereafter referred to as the "flip function"), is esesential for Othello software. The efficiency of the flip function significantly influences the overall computation speed of the search algorithm. In Edax, an array of flip functions is implemented, allowing the appropriate function to be selected via compilation options. It is worth noting that all of the flip functions implemented in Edax require index access to precomputed tables (except for the most naïve implementation using slow for-loops), making them unsuitable for direct transcription into SMT solvers.

```
int bit_count(const uint64_t b)
{
    uint64_t c = b
        - ((b >> 1) & 0x7777777777777777ULL)
        - ((b >> 2) & 0x3333333333333333ULL)
        - ((b >> 3) & 0x1111111111111111ULL);
    c = ((c + (c >> 4)) & 0x0F0F0F0F0F0F0F0FULL) * 0x0101010101010101ULL;

    return  (int)(c >> 56);
}
```

Figure 3. A C++ implementation of a function to population count. This is based on Edax 4.4 (Delorme, 2021) with modifications by the author. Edax 4.4 is licenced under GNU General Public License v3.0 (Free Software Foundation, 2007).

As shown in Figure 4, We found an algorithm in a separate GitHub repository (primenumber, 2016) that does not use index access to tables at all. This can be easily transcribed for use with SMT solvers. The code shown in Figure 4 is also available at GitHub: https://github.com/eukaryo/reversi33 .)

Definition of the Othello Variant

We defined the rules for the Othello variant as follows.

- The game does not end even in the case of two consecutive passes.
- The game ends after the 42nd move, including passes.
- The value of the game record equals to the maximum number of legal moves in all observed board positions, including the final position at game's end.
- Excluding the above stipulations, the rules align with those of traditional Othello.

Given the above definition, for $n = 33$ and $n = 34$, we can assert that a game record with a value of n or greater exists if and only if there is a position that is reachable from the initial position and offers n legal moves. The reasons for this are as follows.

As a preliminary experiment, we conducted an exhaustive search during the opening phase and confirmed that up to 2 passes can occur by the time the number of stones on the board increases from 17 to 18. Consequently, the maximum number of passes that can occur while the number of stones on the board increases from 30 to 31 is 15.

In order to achieve 33 legal moves, the total number of stones must not exceed 31. Consequently, the value must reach 33 at the latest immediately following a pass after the placement of the 31st stone. As such, consideration of 42 moves (comprising 26 stone placements, a maximum of 15 interspersed passes, and a final pass) is sufficient. Similarly, for cases where there are 34 legal moves, consideration of 42 moves is sufficient.

RESULTS

Proven to be 34 Under Unrestricted Conditions

Initially, we disregarded the constraint regarding whether a position is reachable from the initial position. Assuming that the central four squares (namely D4, D5, E4, and E5) always contain stones, we proceeded to derive a proof for the maximum number of legal moves.

We used SMT solver z3 (De Moura and Bjørner, 2008) version 4.12.2 for our proof. In z3, it is possible to perform arithmetic operations on bitvectors, as well as bit manipulation and logical shifts. To input the problem into z3, the function to generate moves and the function to perform population count (described in Methods section above) were implemented using only the operations available in z3.

As a result, the maximum number of legal moves was determined to be 34. We proved that it is unsatisfiable to have 35 or more legal moves, regardless of the number of empty squares. This value can

```
__m256i flipVertical(const __m256i dbd) {
    const __m256i byteswap_table = _mm256_set_epi8(
        8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7
    );
    return _mm256_shuffle_epi8(dbd, byteswap_table);
}
__m256i upperbit(__m256i p) {
    p = _mm256_or_si256(p, _mm256_srli_epi64(p, 1));
    p = _mm256_or_si256(p, _mm256_srli_epi64(p, 2));
    p = _mm256_or_si256(p, _mm256_srli_epi64(p, 4));
    p = _mm256_andnot_si256(_mm256_srli_epi64(p, 1), p);
    p = flipVertical(p);
    p = _mm256_and_si256(p, _mm256_sub_epi64(_mm256_setzero_si256(), p));
    return flipVertical(p);
}
uint64_t hor(const __m256i lhs) {
    __m128i lhs_xz_yw = _mm_or_si128(_mm256_castsi256_si128(lhs), _mm256_extractf128_si256(lhs, 1));
    return uint64_t(_mm_cvtsi128_si64(lhs_xz_yw) | _mm_extract_epi64(lhs_xz_yw, 1));
}
uint64_t flip(const uint64_t player, const uint64_t opponent, const int pos) {
    const __m256i pppp = _mm256_set1_epi64x(player);
    const __m256i oooo = _mm256_set1_epi64x(opponent);
    const __m256i OM = _mm256_and_si256(oooo, _mm256_set_epi64x(
        0xFFFFFFFFFFFFFFFFULL, 0x7E7E7E7E7E7E7E7EULL, 0x7E7E7E7E7E7E7E7EULL, 0x7E7E7E7E7E7E7E7EULL));
    __m256i flipped, outflank, mask;
    mask = _mm256_srli_epi64(_mm256_set_epi64x(
        0x0080808080808080ULL, 0x7F0000000000000ULL, 0x0102040810204000ULL, 0x0040201008040201ULL), 63 - pos);
    outflank = _mm256_and_si256(upperbit(_mm256_andnot_si256(OM, mask)), pppp);
    flipped = _mm256_and_si256(_mm256_slli_epi64(_mm256_sub_epi64(_mm256_setzero_si256(), outflank), 1), mask);
    mask = _mm256_slli_epi64(_mm256_set_epi64x(
        0x0101010101010100ULL, 0x00000000000000FEULL, 0x0002040810204080ULL, 0x8040201008040200ULL), pos);
    outflank = _mm256_and_si256(_mm256_and_si256(mask, _mm256_add_epi64(_mm256_or_si256(
        OM, _mm256_andnot_si256(mask, _mm256_set1_epi8(0xFF))), _mm256_set1_epi64x(1))), pppp);
    flipped = _mm256_or_si256(flipped, _mm256_and_si256(_mm256_sub_epi64(outflank, _mm256_add_epi64(
        _mm256_cmpeq_epi64(outflank, _mm256_setzero_si256()), _mm256_set1_epi64x(1))), mask));
    return hor(flipped);
}
```

Figure 4. A C++ implementation of the flip function (using Intel intrinsics for AVX2). This is based on issen (primenumber, 2016) with modifications by the author. The GitHub repository is licenced under GNU General Public License v3.0 (Free Software Foundation, 2007).

be considered as the upper bound for the maximum number of legal moves that can be achieved from the initial position through a series of legal moves.

Another discovery was that among the positions with 33 legal moves, the maximum number of empty squares was 38 (for example, in the position shown in Figure 5a). For positions with 34 legal moves, the maximum was 37 (for example, in the position shown in Figure 5b). We proved that if a value exceeding these maximums is given as a constraint, it becomes unsatisfiable. (As declared in the data availability section below, the source codes and the outputs of analyses in this subsection are available at GitHub: https://github.com/eukaryo/reversi33 .)

The Generated Positions Were Found to Be Unreachable from the Initial Position

Next, we investigated whether the generated positions could be legally reached from the initial position. We implemented a function that takes a position as input and enumerates all positions that could directly precede the given position, followed by conducting a depth-first search.

To enhance computational efficiency, we pre-enumerated all positions reachable from the initial position that contain 15 stones and have at least one legal move (19,785,690 variations when considering symmetric positions as identical), storing them in a hash table. When the aforementioned backward depth-first search arrives at a position with 15 stones, we can conclude that the original position is reachable from the initial position if and only if that position exists in the hash table.

It is worth noting that a pruning technique can be applied to the aforementioned backward depth-first search. If a position is reachable from the initial position, the graph formed by treating the stones as nodes and connecting stones that exist within a Moore neighborhood (i.e., differences in x-coordinate and y-coordinate are either one or zero) will always yield a connected graph. Therefore, during the aforementioned backward depth-first search, pruning can be performed immediately once such a graph becomes disconnected.

As a result, we determined that the generated positions were unreachable from the initial position. By iteratively adding a constraint that prevented the generation of identical positions, we produced over 1000

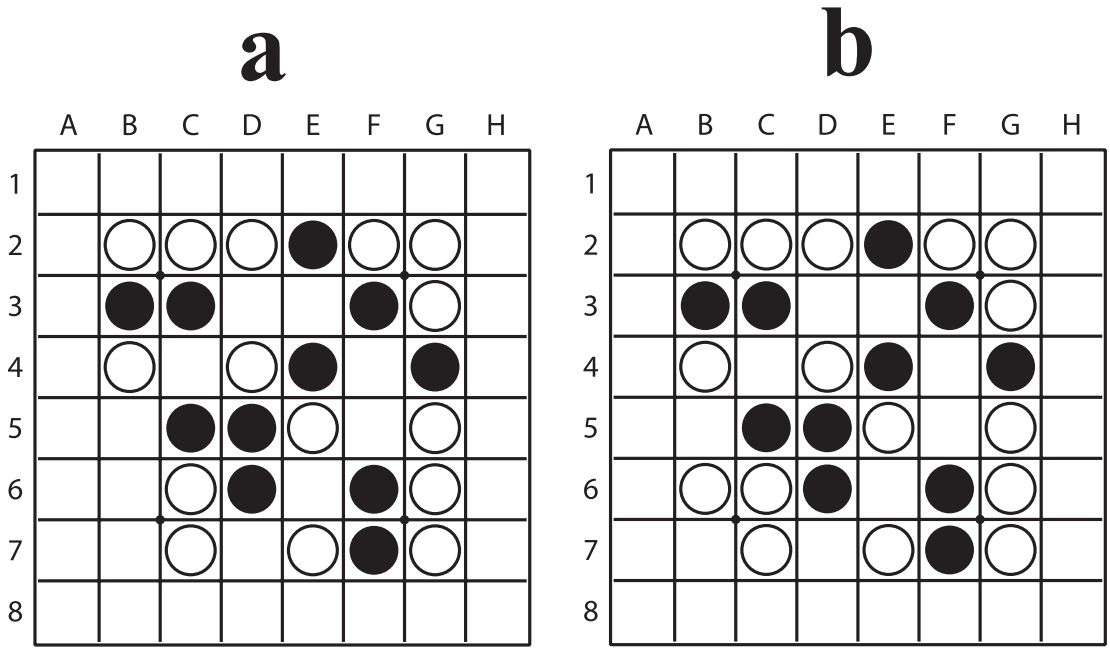

Figure 5. Positions with 33 or more legal moves that the SMT solver z3 outputted, without considering whether they are reachable from the initial position. Both of the following are positions with black to move. (a) A position with 33 legal moves and 38 empty squares. (b) A position with 34 legal moves and 37 empty squares.

positions, none of which were reachable from the initial position (data not shown).

Conversion of Problems to SAT Solver-Compatible Format and Their Resolution

Building on the results above, we sought to enable the SAT solver to simultaneously identify positions reachable from the initial position that feature 33 or more legal moves, along with the corresponding game records. As a preparatory step, we defined the rules for the Othello variant as described in Methods section.

First, we implemented the entire aforementioned Othello variant for the z3 SMT solver. However, the solution did not complete within 24 hours. Furthermore, to the best of our knowledge, z3 does not have a feature that permits the export of problem input in a format compatible with an external SAT solver, nor does it allow for the input of solutions provided by an external SAT solver.

Therefore, we developed software that, given a problem setting, converting it into Conjunctive Normal Form (CNF) which is equivalent to the original problem in terms of satisfiability, and outputs it in DIMACS CNF format. In addition, for the case that the solution was found to be satisfiable, we also developed software to reconstruct the game record from the assignment of Boolean values to variables of the original problem. This approach enabled us to solve the aforementioned Othello variant using the latest SAT solvers.

We utilized the SAT solver kissat (Biere and Fleury, 2022), which demonstrated superior performance in the SAT Competition 2022, to solve the problem. The solver features a “target” option. It accepts the assumption whether the solution is satisfiable or unsatisfiable, and the solver will solve the problem efficiently if the assumption is correct; even if the assumption is incorrect, it will still reach the correct conclusion, albeit potentially taking more time than when not using this option. In this experiment, both assumptions were tried in parallel on a Ryzen 5950X CPU.

As a result, we obtained a position where there were 33 legal moves (as shown in Figure 1b) and the game record from the initial position. When we imposed a constraint that the number of legal moves must be 34 or more, we found this constraint to be unsatisfiable. Additionally, we introduced an extra rule to the aforementioned Othello variant that only considers the game record when it’s Black’s turn to move. As a result, we obtained a position where there were 33 legal moves for Black (as shown in Figure 1a) and the game record from the initial position. All three solutions were acquired within a 24-hour timeframe.

(As declared in the data availability section below, the source codes and the outputs of analyses in this subsection are available at GitHub: https://github.com/eukaryo/reversi33 .)

DISCUSSION

In this study, we proved that among the positions reachable from the initial position in Othello, there exists a position with 33 legal moves, and that 33 is the maximum number of legal moves. Additionally, we also proved that, without considering whether the position is reachable from the initial position, the maximum number of legal moves increases to 34. This study not only elucidates of one aspect of Othello but also provides valuable information for the development of memory-safe and robust Othello software.

In the course of this study, we successfully constructed positions that fulfilled the specified conditions, thereby demonstrating that the maximum numbers act as a lower bound. Conversely, we verified that these maximum numbers also act as a tight upper bound by demonstrating that an increment by one results in the SAT solver returning "unsatisfiable". To validate this proof, it is crucial to ensure the absence of bugs both in the code we developed to convert the Othello problem to CNF for the SAT solver and within the SAT solver itself. Although we carefully checked our code for bugs, the potential introduction of formal verification to heighten certainty may be considered. This represents a limitation of our study. However, the primary aim of our study was to determine the existence of a position that is reachable from the initial state and in which the number of legal moves exceeds 32. Therefore, this was deemed beyond the scope of our study.

CONCLUSIONS

This study presented a scenario where a problem that proved intractable with the SMT solver z3 was solved within a reasonable time frame by converting it to CNF format and utilizing the latest SAT solver kissat (Biere and Fleury, 2022). Given the rapid advancements in SAT solvers through annual competitions, this case appears to support a perspective that challenges the design of SMT solvers closely integrated with older SAT solvers. However, further research is warranted on this matter.

We hope this research proves beneficial to readers endeavoring to develop Othello software, understand the nature of Othello and other games, and those working towards the enhancement of SAT solvers and other theoretical solvers.

ADDITIONAL INFORMATION AND DECLARATIONS

Competing Interests

The author declares that there are no competing interests.

Author Contributions

Hiroki Takizawa conceived and designed the research, implemented and performed the computational experiments, analyzed the data, prepared figures, authored drafts of the paper, and approved the final draft.

Data Availability

The source codes and the outputs of analyses are available at GitHub: https://github.com/eukaryo/reversi33 .

Funding

The author has received no funding for this study.

REFERENCES

Abramson, B. (1989). Control strategies for two-player games. *ACM Comput. Surv.*, 21(2):137–161.

Biere, A. and Fleury, M. (2022). Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In Balyo, T., Heule, M., Iser, M., Järvisalo, M., and Suda, M., editors, *Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions*, volume B-2022-1 of *Department of Computer Science Series of Publications B*, pages 10–11. University of Helsinki.

Buro, M. (1999). How machines have rearnea to play othello. *IEEE intelligent systems & their applications*, 14(6):12–14.

Campbell, M., Hoane, A., and hsiung Hsu, F. (2002). Deep blue. *Artificial Intelligence*, 134(1):57–83.

De Moura, L. and Bjørner, N. (2008). Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Delorme, R. (2021). edax-reversi. https://github.com/abulmo/edax-reversi. Last retrieved 2023-07-07.

Free Software Foundation, I. (2007). Gnu general public license, version 3. http://www.gnu.org/licenses/gpl.html. Last retrieved 2020-01-01.

Kaneko, T. and Takizawa, T. (2019). Computer shogi tournaments and techniques. *IEEE Transactions on Games*, 11(3):267–274.

primenumber (2016). issen/src/move_generator.cpp. https://github.com/primenumber/issen/blob/72f450256878094ffe90b75f8674599e6869c238/src/move_generator.cpp. Last retrieved 2023-07-07.

Reingold, E. M., Nievergelt, J., and Deo, N. (1977). *Combinatorial algorithms: theory and practice*. Prentice Hall College Div.

Schaeffer, J., Lake, R., Lu, P., and Bryant, M. (1996). Chinook the world man-machine checkers champion. *AI magazine*, 17(1):21–21.

Shannon, C. E. (1950). Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.