Combining Combinatorial Game Theory with an α-β Solver for Clobber: Theory and Experiments

Jos W.H.M. Uiterwijk(✉) and Janis Griebel

Department of Data Science and Knowledge Engineering,
Maastricht University, Maastricht, The Netherlands
uiterwijk@maastrichtuniversity.nl, janis.griebel@online.de

Abstract. Combinatorial games are a special category of games sharing the property that the winner is by definition the last player able to move. To solve such games two main methods are being applied. The first is a general α-β search with many possible enhancements. This technique is applicable to every game, mainly limited by the size of the game due to the exponential explosion of the solution tree. The second way is to use techniques from Combinatorial Game Theory (CGT), with very precise CGT values for (subgames of) combinatorial games. This method is only applicable to relatively small (sub)games. In this paper, which is an extended version of [7], we show that the methods can be combined in a fruitful way by incorporating an endgame database filled with CGT values into an α-β solver.

We apply this technique to the game of Clobber, a well-known all-small combinatorial game. Our test suite consists of 20 boards with sizes up to 18 squares. An endgame database was created for all subgames of size 8 and less. The CGT values were calculated using the CGSUITE package. Experiments reveal reductions of at least 75% in number of nodes investigated.

1 Introduction

Clobber is a two-player game invented by Albert, Grossman and Nowakowski in 2001 [1]. It is played on a rectangular board, the size of which can be varied. At the start the squares are alternately filled with black and white stones. The first player (Black, also called Left) controls the black stones, the second player (White, also called Right) controls the white stones. Commonly the start position is a filled 8×8 board as shown in Fig. 1.

The players move in alternating order. A move is done by picking one of the own stones and moving it to an orthogonally adjacent square that is occupied by an opponent stone. The opponent stone gets removed from the board ("clobbered"). The player that makes the last move wins. An example move is shown in Fig. 2, where Black has opted to clobber White's stone in the upper left corner.

T. Bosse and B. Bredeweg (Eds.): BNAIC 2016, CCIS 765, pp. 78–92, 2017.
DOI: 10.1007/978-3-319-67468-1_6

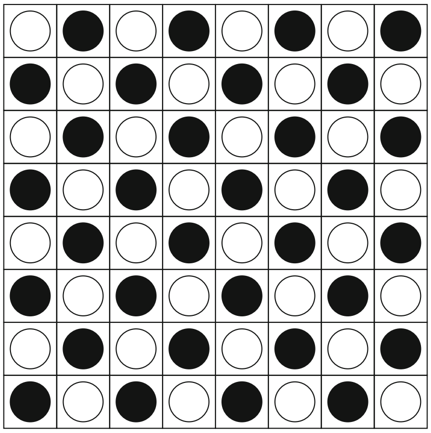


Fig. 1. Common Clobber start position.

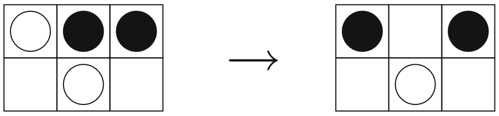


Fig. 2. Illustration of a move in Clobber.

Important for Clobber are the following facts:

– Clobber is a 2-player perfect-information, deterministic game.
– Clobber is a converging game. With each move the total number of stones gets decreased by one.
– Clobber is a partizan game. Left and Right have their own stones, which results in the fact, that each player has its own moves.
– If a player can make a move in a specific position, the opponent player could also make a move in that position, because then two opponent stones are orthogonally adjacent. This means the game is a so-called *all-small* game [2].
– The symmetry of a position does not influence the outcome. If the board is mirrored vertically and/or horizontally, or rotated by a multiple of 90°, the outcome of the position does not change.
– No draws are possible: either the first player (Left) or the second player (Right) wins by making the last move.

As a result of these facts Clobber belongs to the category of *combinatorial games*, for which a whole theory has been developed, the Combinatorial Game Theory (CGT). In this article we describe the results of building an endgame database with exact CGT values for the game of Clobber. Section 2 gives an overview of related research. In Sect. 3 we give an introduction to the Combinatorial Game Theory, as far as applicable to Clobber. In Sect. 4 we describe the methods implemented. Next, in Sect. 5, we show our experiments. In Sect. 6 we give our conclusions and some suggestions for further research.

2 Related Work

The literature regarding CGT for Clobber is scarce, except the paper introducing the game [1]. Claessen constructed for Clobber (partly filled) databases with exact CGT values, but they were not used to solve boards, but to enhance a Clobber playing engine based on the MCTS framework [5].

Other related work has been done by Müller [9] who applied CGT values to solve local endgames in Go. However, the global search is not an α-β search, and CGT values are not obtained from CGT endgame databases, but calculated on the spot. Müller and Li [11] showed results for combining α-β search with CGT pruning and ordering. Their games are artificial games with very special properties, having nothing in common with "real" combinatorial games. No endgame databases were used.

Most other related work has been done in the area of the game Amazons. Müller [10] used CGT to establish bounds in a specialized divide-and-conquer approach. He was able to solve 5×5 Amazons. No CGT endgame databases were used. Snatzke [13] built CGT endgame databases for a very restricted version of Amazons, namely for subgames fitting on a 2×11 board with exactly 1 queen per player. This was extended in [14] with new results for some small databases of other shapes, with 1 to 4 queens. He did not incorporate the use of his databases in a general Amazons solver. Tegos [16] was the first to combine endgame databases for Amazons in an α-β-based Amazons playing program. Besides (traditional) minimax endgame databases (without CGT information) he also implemented CGT endgame databases. These contained just thermograph information, not precise CGT values, and therefore only could be used for (heuristic) move-ordering purposes. Recently, Song [15] implemented endgame databases in an Amazons solver. Again, the databases did only contain heuristic (thermograph) information useful for narrowing the bounds in the solving process. With his program 5×6 Amazons has been solved.

As far as we know the only other research reporting on combining global α-β searches with endgame databases with precise CGT values is our related work on Domineering [3,17]. In these articles we showed that equipping a simple α-β solver with CGT endgame databases gives reductions up to 99% in number of nodes investigated for solving a testset of 36 non-trivial rectangular boards.

3 Combinatorial Game Theory for Clobber

In this section we give a short introduction to the Combinatorial Game Theory as far as relevant for Clobber. For a more thorough introduction, we refer to the literature, in particular [2,4,6].

In a combinatorial game, the players are conventionally called Left and Right. For Clobber, Left is the player moving the black stones, therefore also denoted by Black, and similarly Right (White) moves the white stones.

In CGT a game G is represented by its left and right *options* G^L and G^R, so $G = \{G^L | G^R\}$. In this representation, G^L and G^R stand for sets of games (the options) that players Left and Right, respectively, can reach by making one move in the game. The *value* of a game indicates how good a game is for a player, where positive values indicate an advantage for Left and negative values an advantage for Right. There are several types of values for Clobber positions. These are treated in the next subsections.

3.1 Numbers

Numbers have the property that any option is a number itself, and that no left option has a higher value than any right option. The simplest number game is the endgame $\{|\}$, denoted as 0. In this position, no player has any available moves, so it is a loss for the player to move. In Clobber, a position with just one stone, either black or white, is an endgame position, with value 0.

Larger or smaller numbers are built recursively:

$$\begin{aligned}
0 &= \{|\} \\
1 &= \{0|\} = \{\{|\}|\} \\
2 &= \{1|\} = \{\{0|\}|\} = \{\{\{|\}\,|\}|\} \\
-1 &= \{|0\} = \{|\{|\}\} \\
-2 &= \{|-1\} = \{|\{|0\}\} = \{|\{|\,\{|\}\}\}
\end{aligned}$$

Also fractions are possible. However, Clobber has the property that no number games are possible, except the endgame 0.

3.2 Nimbers

Nimbers are a class of games where the first player to move wins. The simplest such game is called Star or $*$, defined as $* = \{0|0\} = \{\{|\}\,|\,\{|\}\}$, where the player to move has just one option, leading to the endgame. An example is the simple Clobber game of Fig. 3.

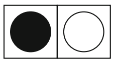


Fig. 3. A $*$ position in Clobber.

A *nimber* is a game defined as follows:

$$\begin{aligned}
*0 &= 0 \\
*1 &= * \\
*2 &= \{\{*0, *1\}|\{*0, *1\}\} \\
*3 &= \{\{*0, *1, *2\}|\{*0, *1, *2\}\} \\
*n &= \{\{*0, *1, ..., *(n-1)\}|\{*0, *1, ..., *(n-1)\}\}
\end{aligned}$$

Note that $*0 = 0$ is the only game being both a number and a nimber. All other nimbers have the property that they are a win for the first player to move.

3.3 Ups and Downs

Ups and *downs* are other types of values. An up or $\uparrow$ is defined as $\uparrow = \{0|*\}$ and is strictly positive, meaning that Left wins this game, irrespective of who starts. Down or $\downarrow$ is its negative, defined as $\downarrow = -\uparrow = \{*|0\}$ and is strictly negative, meaning that Right wins this game, irrespective of who starts. See Fig. 4 for an example $\uparrow$ (left) and $\downarrow$ (right) position in Clobber.

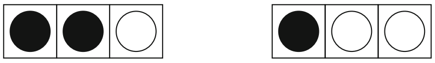


Fig. 4. An up and a down position in Clobber.

Next, in Clobber often positions occur that are in fact sums of ups and downs. For these, special notations are introduced, where $\uparrow*$ means $\uparrow + *$, etc.:

$$\begin{aligned} \uparrow &= \{0|*\} & \downarrow &= \{*|0\} \\ \uparrow + \uparrow = \Uparrow &= \{0|\uparrow*\} & \downarrow + \downarrow = \Downarrow &= \{\downarrow*|0\} \\ \uparrow + \uparrow + \uparrow = \Uparrow\!\!\!\uparrow &= \{0|\Uparrow*\} & \downarrow + \downarrow + \downarrow = \Downarrow\!\!\!\downarrow &= \{\Downarrow*|0\} \end{aligned}$$

In general (where $n \cdot \uparrow*$ is parsed as $(n \cdot \uparrow)*$, and similarly for $n \cdot \downarrow*$):

$$\begin{aligned} n \cdot \uparrow &= \{0|*\} && \text{if } n = 1 & n \cdot \downarrow &= \{*|0\} && \text{if } n = 1 \\ n \cdot \uparrow &= \{0|(n-1) \cdot \uparrow*\} && \text{if } n > 1 & n \cdot \downarrow &= \{(n-1) \cdot \downarrow*|0\} && \text{if } n > 1 \end{aligned}$$

They are often combined with a *, giving

$$\begin{aligned} \uparrow* &= \{0, *|0\} & \downarrow* &= \{0|0, *\} \\ \Uparrow* &= \{0|\uparrow\} & \Downarrow* &= \{\downarrow|0\} \\ \Uparrow\!\!\!\uparrow* &= \{0|\Uparrow\} & \Downarrow\!\!\!\downarrow* &= \{\Downarrow|0\} \end{aligned}$$

In general:

$$\begin{aligned} n \cdot \uparrow* &= \{0, *|0\} && \text{if } n = 1 & n \cdot \downarrow* &= \{0|0, *\} && \text{if } n = 1 \\ n \cdot \uparrow* &= \{0|(n-1) \cdot \uparrow\} && \text{if } n > 1 & n \cdot \downarrow* &= \{(n-1) \cdot \downarrow|0\} && \text{if } n > 1 \end{aligned}$$

Figure 5 shows a $\Uparrow*$ (left) and a $\Uparrow\!\!\!\uparrow$ (right) position in Clobber.

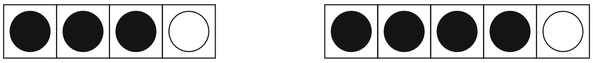


Fig. 5. Examples of multiple up positions in Clobber.

3.4 Infinitesimal and All-Small Games

Definition 1. *A game G is* infinitesimal *if $-x < G < x$ for every positive number x.*

The number 0 and all nimbers, ups and downs are infinitesimal.

Definition 2. *A game G is* all-small *if for every position H in G it holds that Left can move in H if and only if Right can.*

This means that either $G = 0$ or G^L and G^R are both non-empty and all options are all-small. As a consequence, an all-small game is infinitesimal.

Since Clobber has the property that in every position Black can move if and only if White can, it follows that Clobber is an all-small game. As a consequence all Clobber positions are infinitesimal. So, the only number that can occur in Clobber is 0 (the only number that is also a nimber). Furthermore, only nimbers, ups and downs, and combinations thereof can occur in Clobber.

3.5 Canonical Forms

A game in CGT always has a unique *canonical form*. The canonical form describes the *smallest* game G' of a game G where G' is equivalent to G.

The reduction to the canonical form can be obtained considering two conditions. G' is said to be in canonical form if G' has no *dominated options* and when *reversible options* are *bypassed*. In order to reduce a game to its canonical form, the dominated or reversible options should be identified and removed or bypassed, respectively.

An option is dominated when there is at least one other option equal to or better than the option considered. Then the player has a better or at least an equal alternative. More precisely, a left option L_1 is dominated if there exists another left option L_2 with $L_1 \leq L_2$. A right option R_1 is dominated if there exists another right option R_2 with $R_1 \geq R_2$. A game with a dominated option is equivalent to the smaller game G' with this option removed.

An example Clobber position with dominated options is given in Fig. 6.

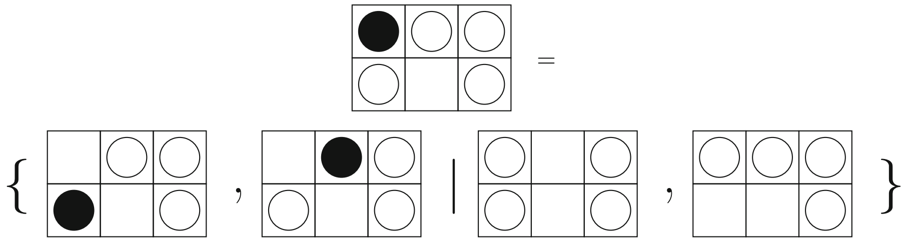
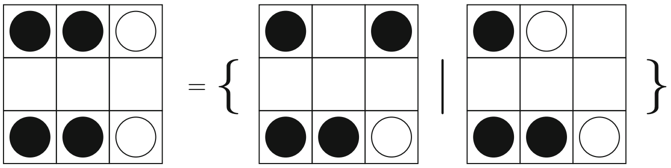


Fig. 6. A Clobber position with dominated options.

The left options have values 0 and $\downarrow$, respectively, the right options both have value 0. Therefore, this position has value $\{0, \downarrow \mid 0, 0\}$. Since 0 dominates $\downarrow$ for Left and 0 dominates 0 for Right, the canonical form of this game is $\{0 \mid 0\} = *$.

An option is reversible if the resulting game contains an option for the next player which is, from its point of view, equal to or better than the current game. In this case it is obvious that the next player would perform this equal or better option immediately after the reversible option. We can then in fact directly use the options of this resulting position instead. We call this *bypassing a reversible option*. More precisely, a left option L_1 of a game G is reversible if L_1 contains a right option $L_1^{R_1}$ with $L_1^{R_1} \leq G$. L_1 can then be replaced by the left options of $L_1^{R_1}$. Vice versa, a right option R_1 of a game G is reversible if R_1 contains a left option $R_1^{L_1}$ with $R_1^{L_1} \geq G$. R_1 can then be replaced by the right options of $R_1^{L_1}$. A game with a reversible option is equivalent to the smaller game G' with this option bypassed.

An example Clobber position with a reversible option is given in Fig. 7.



$$= \{ \uparrow \mid \uparrow * \} = \{ \{ 0 \mid * \} \mid \uparrow * \} = \{ \{ 0 \mid \{ 0 \mid 0 \} \} \mid \uparrow * \} = \{ 0 \mid \uparrow * \}$$

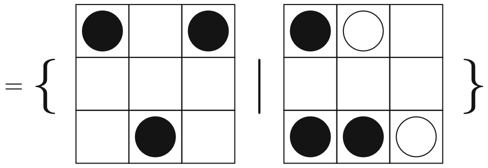


Fig. 7. A Clobber position with a reversible option.

In this position we first have removed the dominated options for both players (moves in the lower row being equivalent with moves in the upper row). Next, the left option $\uparrow$ has as only right option a $*$ position and since $*$ definitely is better for Right than the initial position $(\uparrow + \uparrow)$, we may replace the left option in the initial position by the left option of $*$, being 0.

The canonical form helps to show equivalence of games. Since the canonical form of a game is unique, two games are equivalent if their canonical forms are identical. This paper will deal with canonical forms in the endgame database, explained in Sect. 4.

3.6 Sums of Games

Many combinatorial games have the nice property that they can split into subgames that do not interact. Clobber belongs to this category. When this happens we say that the value of the position is the sum of the values of the subgames. Formally: $G + H = \{G^L + H, G + H^L \mid G^R + H, G + H^R\}$, which states that a player can choose the subgame in which to play.

When the subgames have simple values (0, nimbers, ups and downs) we then can easily add the resulting values. Here a 0 acts as an identity $(0 + G = G)$. Nimbers are added using Nim-addition $(*m + *n = *(m \oplus n))$, where $\oplus$ is the exclusive-or operator applied to the binary representations of the operands. Ups and downs are summed using the rules $m \cdot \uparrow + n \cdot \uparrow = (m + n) \cdot \uparrow$ and $\downarrow = -1 \cdot \uparrow$.

As an example, consider the Clobber position in Fig. 8.

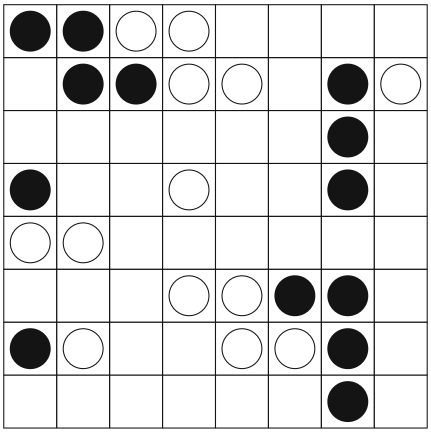


Fig. 8. An example Clobber position with 6 subgames.

The subgames in this example position, from top to bottom and left to right, have values $*2$, $\Uparrow*$, $\downarrow$, 0, $*$, and $*2$. Therefore, the value of the whole position is $*2 + \Uparrow* + \downarrow + 0 + * + *2 = \uparrow$. It means that the position is won by Black, regardless of who is starting the game.

4 Methods

In this section we describe the methods implemented in our Clobber solver. The core is a straightforward α-β solver, described in Sect. 4.1. The main enhancement consists of the construction of the CGT endgame database (Sect. 4.2). Section 4.3 shows how the CGT values from the endgame database can be used within the α-β framework.

4.1 Basic α-β Solver

For the basic Clobber solver we implemented a straightforward zero-window α-β [8] searcher. This searcher investigates lines until the end (returning that either Left or Right made the last move and so wins the game). Since our goal was to measure the impact of using CGT on solution tree sizes, we refrained from incorporating any further enhancements to the α-β framework.

4.2 A CGT Database for Clobber

In order to use CGT values for Clobber subgames we have built a Clobber endgame database with all subgames consisting of 1–8 connected stones with arbitrary shape. Positions with mirror symmetry (left/right and/or top/bottom) are unified into a single entry.

The database is used to store pre-calculated CGT values of specific positions. These CGT values are used to get knowledge about its subgames. If a position is reached that only consists of pre-calculated subgames, the whole position can be solved by combining the CGT values using game-logic rules (when the CGT values are all simple).

As a preparation a database must be designed, created and filled with the appropriate values. These steps are described in the following subsections.

Database design
In the database the CGT values must be stored together with the appropriate game positions. The easiest way to store the CGT values is using a character string because of the different kinds of CGT value types. For the game positions a hash code is generated that allows to rebuild the exact positions. The format of the hash code is described in the next subsection.

Since the information that is needed is simple, only one table is needed for the database. The hash code is used as the primary key. Since the hash code consists of two values, a string and an integer for the position width, the primary key is a combined one. A third column is used for the CGT value of the position.

Representing a position
A game position is represented in a hash code to get stored in the database. It is important that the hash code is unique for a position. There are different approaches that can be used to generate such a unique hash code.

In our first approach the board was iterated sequentially and indicated in the hash code. The squares are visited from left to right through all rows from top to bottom. An empty square is indicated by the character “0”, a square with a black stone by “1” and a square with a white stone by “2”. These characters are concatenated to a string and used as the hash code for the position. An example is shown in Fig. 9 (left).

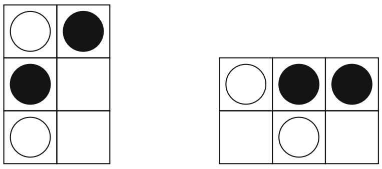


Fig. 9. Two example positions with the same hash code and different values.

This position with value $\{*, \uparrow | *, \downarrow\}$ has hash code “211020”. A problem with a hash code in this format is that it does not include the position size. The position in Fig. 9 (right) is clearly different, with value $\{0|\downarrow\} = *$, but has the same hash code. To solve this issue the position size needs to be stored together with the hash code. For that the width of a position is added as a column to the database. The primary key consists of the combination of the generated hash code and the position width.

Calculating CGT values

Since the database will be used as a lookup for solved positions, it must be filled with CGT values. The CGT values for the Clobber subgames are calculated with the CGSUITE software tool [12]. These values are in canonical form, which makes them a unique representation of the CGT values. They are stored in the third column in the database.

Generating positions

To fill the database a set of positions is needed for which the appropriate CGT values are calculated and stored. To generate the hash codes for these positions an algorithm is implemented that iterates through all possibilities.

The algorithm gets a limit that consists of a number of connected nonempty squares. All possible formations and fillings of the connected squares are iterated to fill the database. To get all positions with n connected nonempty squares, an empty $n \times n$ board is created. In a naive approach, the algorithm would go recursively through all squares, filling each square first with empty, then with a black stone and at last with a white stone. For an $n \times n$ board it would mean that the algorithm generates $3^{(n \times n)}$ combinations of fillings. These include many combinations that exceed the set limit of connected nonempty squares n.

To avoid iterating through combinations that do not fit the set limit n the algorithm counts the number of nonempty squares through the iterations. If the limit n is reached it continues just with empty squares till the last square of the board. The resulting amount of combinations can be described with binomial coefficients. For an $n \times n$ board it means that there are

$$\binom{n \times n}{n} \times 3^n \tag{1}$$

combinations generated.

As a second improvement the algorithm is changed to ignore combinations that contain less than n nonempty squares. Since the algorithm is executed sequentially with an increasing limit n, it does not need to check such combinations. This improvement is obtained by not generating positions that have less than n nonempty squares reaching the last square of the board. This improvement leads to a smaller amount of evaluated combinations that can be described by the following formula, since always exactly n squares are nonempty:

$$\binom{n \times n}{n} \times 2^n \tag{2}$$

Note that we only consider positions that do not consist of subgames, since it would mean that the amount of nonempty squares is less than n for each subgame separately and therefore will have been generated in earlier iterations.

As a last improvement the amount of iterated positions is drastically decreased. Consider that an $n \times n$ board is used to get all formations of n nonempty connected squares. The width n of the position is only needed if all nonempty squares are in a row. Therefore a height of 1 is sufficient to obtain all possible formations, because there is only one line with all nonempty squares in a row. If one square gets occupied that is not in a row with the others, a smaller position with size $(n-1) \times 2$ can be used to generate all missing positions. For a next square that is not in row 1 or 2 it means that a position with size $(n-2) \times 3$ fits to get all possible positions with a maximum of $n-2$ squares in a row, etc. As an example of generating all possible formations for $n = 5$ connected squares, Fig. 10 shows the required board sizes, where the gray stones just indicate example fillings with five stones, irrespective of their actual colors.

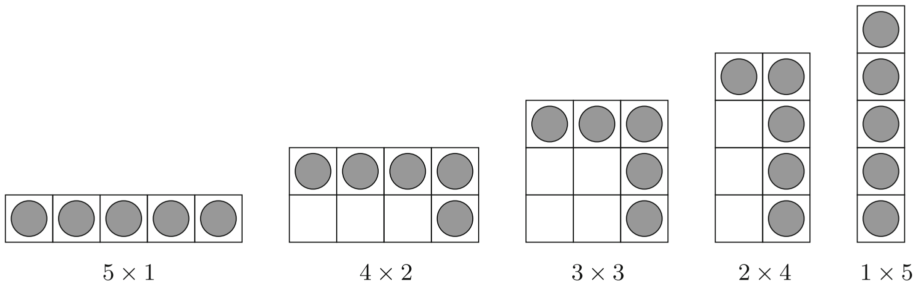


Fig. 10. Board sizes needed to get all possible formations for $n = 5$ connected stones.

We replace the amount of squares, that are not in a row with any other square, but still connected with the other ones, by a variable p. Using boards with size $(n-p+1) \times p$, where p runs from 1 to n, we can generate all possible formations of n connected nonempty squares. The following formula describes the amount of iterated positions for one specific p:

$$\binom{(n-p+1) \times p}{n} \times 2^n$$

When we let p run from 1 to n and calculate the sum of the amount of all formation possibilities, the number of iterated positions is described by the following formula:

$$\left(\sum_{p=1}^{n} \binom{(n-p+1) \times p}{n} \right) \times 2^n \tag{3}$$

To show the improvement by these approaches for the generation of the positions, the amount of iterated positions is listed in Table 1.

The final improvement results in a serious redundancy reduction. Generating one position means that the hash code is generated, a lookup in the database is

Table 1. Improving position generation.

n	No optimization $3^{(n \times n)}$	n chosen squares Eq. (1)	n nonempty squares Eq. (2)	Final Eq. (3)
1	3	3	2	2
2	81	54	24	8
3	19,683	2,268	672	48
4	4.30×10^{7}	1.47×10^{5}	29,120	512
5	8.47×10^{11}	1.29×10^{7}	1.70×10^{6}	7,680
6	1.50×10^{17}	1.42×10^{9}	1.24×10^{8}	1.45×10^{5}
7	2.39×10^{23}	1.88×10^{11}	1.10×10^{10}	3.31×10^{6}
8	3.43×10^{30}	2.90×10^{13}	1.13×10^{12}	8.84×10^{7}

made, and sometimes a new entry is stored in the database. Now the algorithm only generates 8.84×10^7 instead of 3.43×10^{30} positions for $n = 8$ nonempty connected squares. This is an improvement by a factor 4.05×10^{22}.

Filling the database

The C# algorithm, that generates the positions was prepared to run with one specific set n. To fill the database it first runs with $n = 1$. After completing the generation of all positions, n is increased by 1 and the algorithm is started again. The idea is to get positions generated for an n that is as high as possible. The higher the n is the longer the algorithm runs because the amount of iterated positions increases exponentially.

Running on a 3 GHz Quad-Core computer the algorithm needed around 2 h for all positions with $n \leq 7$ nonempty connected squares. For $n = 8$ it took more than 3 days. Expecting a few months for $n = 9$ the generation was stopped. The number of generated entries for n connected nonempty squares is listed in Table 2. Note that for $n \geq 4$ these numbers are smaller than the ones in the right column of Table 1, since the positions generated may consist of unconnected subfragments already in the database for smaller values of n. The database consumes 79.23 MB memory space.

Table 2. Stored positions in the database.

n	1	2	3	4	5	6	7	8
	2	8	48	304	2,016	13,824	97,280	697,600

sum = 811,082 entries

4.3 Combining the CGT Endgame Database with α-β

Combining the CGT endgame database with an α-β solver can be achieved by a simple adaptation of the basic α-β algorithm. As soon as a (sub)game with at most 8 connected stones is encountered, its value is looked up in the database and using this value it is determined whether the (sub)game is a win or loss for the player to move.

As a result, when investigating an $m \times n$ board lines are never searched deeper than $(m \times n) - 8$, when the position is necessarily reduced to 8 stones (since each move consists of removing exactly 1 stone). Moreover, many lines will even terminate at shallower depths when a position splits into multiple subgames of size ≤ 8. As long as there are subgames with more than 8 stones, the investigation continues. As soon as each subgame separately contains at most 8 stones, the search terminates and game logic is used to determine the winner. This ensures determining correctly the game-theoretical result.

5 Experiments and Discussion

In this section the implemented features are tested to generate insight into their impact on improving a plain α-β search. In the setup we used the α-β implementation, without and with database support, as described in the previous section.

Boards with different sizes are investigated. The size runs up to 18 squares with a maximum width of 8. The board is initially filled in the common way as a checkerboard, starting with a black stone. Since the results are not comparable on different computers considering the investigated time, the number of visited nodes is used as the value to compare.

Two remarks are in order. First, although the efficiency of α-β depends on move ordering, we have not incorporated this in the current implementation. It is, however, far from trivial to find a good move-ordering technique for an all-small game like Clobber. Second, who starts influences the amount of visited nodes. Therefore each position is played twice, once with Black and once with White as the player to move. Both amounts of visited nodes are summed and given as the results shown in Table 3.

In this table all investigated board sizes (with the board between brackets) are listed in the left column. The second column shows the number of visited nodes (disregarding the root in the tree) using the default α-β search (denoted α-β_noDB). The third column contains the number of visited nodes when the database is used (α-β_withDB). Column 4 shows the reductions for α-β_withDB compared to α-β_noDB as percentages. Finally, the last column contains the outcome classes for the boards. Here, $\mathcal{P}$ means a win for the previous player, so a loss for the player to move, $\mathcal{N}$ means a win for the next player (the player to move), whereas $\mathcal{L}$ and an $\mathcal{R}$ denote wins for Left (Black) and Right (White), irrespective who starts the game. The results in this column match with the results in [1] in so far as given there.

One should keep in mind that the filling of each board starts with a black stone and then goes further alternating between White and Black. Therefore, in the position of size 1×7, if the filling would start with a white stone, then Black would be determined as the winner, irrespective of who starts (i.e., outcome class $\mathcal{L}$). All other results just depend on the player to move first and do not change.

Table 3. Comparison of numbers of nodes visited during searches.

Board size	α-β_noDB	α-β_withDB	red. (%)	Outcome class
1 (1×1)	0	0	100.00%	$\mathcal{P}$
2 (1×2)	2	0	100.00%	$\mathcal{N}$
3 (1×3)	2	0	100.00%	$\mathcal{N}$
4 (1×4)	3	0	100.00%	$\mathcal{N}$
4 (2×2)	6	0	100.00%	$\mathcal{N}$
5 (1×5)	15	0	100.00%	$\mathcal{P}$
6 (1×6)	42	0	100.00%	$\mathcal{P}$
6 (2×3)	68	0	100.00%	$\mathcal{P}$
7 (1×7)	49	0	100.00%	$\mathcal{R}$
8 (1×8)	60	0	100.00%	$\mathcal{N}$
8 (2×4)	162	0	100.00%	$\mathcal{N}$
9 (3×3)	222	2	99.10%	$\mathcal{N}$
10 (2×5)	654	66	89.91%	$\mathcal{N}$
12 (2×6)	16,150	1,003	93.79%	$\mathcal{P}$
12 (3×4)	19,532	1,412	92.77%	$\mathcal{P}$
14 (2×7)	45,502	2,849	93.74%	$\mathcal{N}$
15 (3×5)	125,638	14,573	88.40%	$\mathcal{N}$
16 (2×8)	304,230	52,384	82.78%	$\mathcal{N}$
16 (4×4)	625,544	105,596	83.12%	$\mathcal{N}$
18 (3×6)	24,626,986	6,222,980	74.73%	$\mathcal{P}$

6 Conclusions and Future Research

We have shown how CGT values of subgames can be used as an enhancement of a basic α-β solver for Clobber. A database with exact CGT values was built for all (sub)games up to 8 connected stones. Reductions depend on board size, going down from 100% for the boards in the database to 75% for the 3×6 board.

As future research we want to extend the database to include larger subgames. Moreover, we see opportunities to incorporate more game-dependent knowledge, like move-ordering heuristics, to solve larger and more complex Clobber boards. Also, well-known game-independent techniques like transposition tables can greatly enhance the solving efficiency. Of course, it has to be investigated for such optimized α-β solvers for Clobber whether incorporating a CGT endgame database gives similar efficiency enhancements. Finally, we envision using CGT theory in solvers for other combinatorial games (besides Domineering [3] and Clobber (this work)) to get more insight into the game conditions for such a fruitful combination.

References

1. Albert, M.H., Grossman, J.P., Nowakowski, R.J., Wolfe, D.: An introduction to Clobber. Integers Electron. J. Comb. Number Theor. **5**(2), 12 p. (2005)
2. Albert, M.H., Nowakowski, R.J., Wolfe, D.: Lessons in Play: An Introduction to Combinatorial Game Theory. A K Peters, Wellesley (2007)
3. Barton, M., Uiterwijk, J.W.H.M.: Combining combinatorial game theory with an α-β solver for Domineering. In: Grootjen, F., Otworowska, M., Kwisthout, J. (eds.), BNAIC 2014: Proceedings of the 26th Benelux Conference on Artificial Intelligence, pp. 9–16. Radboud University, Nijmegen (2014)
4. Berlekamp, E.R., Conway, J.H., Guy, R.K.: Winning Ways for your Mathematical Plays, vol. 1–2. Academic Press, London (1982). 2nd edition, in four volumes: vol. 1 (2001), vols. 2, 3 (2003), vol. 4 (2004), A K Peters. Wellesley
5. Claessen, J.: Combinatorial game theory in Clobber. Master's thesis, Maastricht University (2011)
6. Conway, J.H.: On numbers and games. Academic Press, London (1976). 2nd edition A K Peters, Natick, MA (2001)
7. Griebel, J., Uiterwijk, J.W.H.M.: Combining combinatorial game theory with an α-β solver for Clobber. In: Bosse, T., Bredeweg, B. (eds.), BNAIC 2016: Proceedings of the 28th Benelux Conference on Artificial Intelligence, pp. 48–55. University of Amsterdam/Vrije Universiteit Amsterdam, Amsterdam (2016)
8. Knuth, D.E., Moore, R.W.: An analysis of alpha-beta pruning. Artif. Intell. **6**, 293–326 (1975)
9. Müller, M.: Global and local game tree search. Inf. Sci. **135**, 187–206 (2001)
10. Müller, M.: Solving 5×5 Amazons. In: The 6th Game Programming Workshop (GPW 2001), Hakone (Japan), 2001, vol. 14, IPSJ Symposium Series, pp. 64–71 (2001)
11. Müller, M., Li, Z.: Locally informed global search for sums of combinatorial games. In: van den Herik, H.J., Björnsson, Y., Netanyahu, N.S. (eds.) CG 2004. LNCS, vol. 3846, pp. 273–284. Springer, Heidelberg (2006). doi:10.1007/11674399_19
12. Siegel, A.N.: Combinatorial game suite: a computer algebra system for research in combinatorial game theory (2003). http://cgsuite.sourceforge.net/
13. Snatzke, R.G.: Exhaustive search in the game Amazons. In: Nowakowski, R.J. (ed.) More Games of No Chance, Proceedings of MSRI Workshop on Combinatorial Games, Berkeley, CA, July 2000, MSRI Publ., vol. 42, pp. 261–278. Cambridge University Press, Cambridge (2002)
14. Snatzke, R.G.: New results of exhaustive search in the game Amazons. Theoret. Comput. Sci. **313**, 499–509 (2004)
15. Song, J.: An enhanced solver for the game of Amazons. Master's thesis, University of Alberta (2013)
16. Tegos, T.: Shooting the last arrow. Master's thesis, University of Alberta (2002)
17. Uiterwijk, J.W.H.M., Barton, M.: New results for Domineering from combinatorial game theory endgame databases. Theoret. Comput. Sci. **592**, 72–86 (2015)