Solving Cram Using Combinatorial Game Theory

Jos W. H. M. Uiterwijk(✉)

Department of Data Science and Knowledge Engineering (DKE), Maastricht University, Maastricht, The Netherlands
uiterwijk@maastrichtuniversity.nl

Abstract. In this paper we investigate the board game Cram, which is an impartial combinatorial game, using an $\alpha\beta$ solver. Since Cram is a notoriously hard game in the sense that it is difficult to obtain reliable and useful domain knowledge to use in the search process, we decided to rely on knowledge obtained from Combinatorial Game Theory (CGT).

The first and most effective addition to our solver is to incorporate endgame databases pre-filled with CGT values (nimbers) for all positions fitting on boards with at most 30 squares. This together with two efficient move-ordering heuristics aiming at early splitting positions into fragments fitting in the available databases gives a large improvement of solving power. Next we define five more heuristics based on CGT that can be used to further reduce the sizes of the search trees considerably.

In the final version of our program we were able to solve all odd $\times$ odd Cram boards for which results were available from the literature (even $\times$ even and odd $\times$ even boards are trivially solved). Investigating new boards led to solving two boards not solved before, namely the 3×21 board, a first-player win, and the 5×11 board, a second-player win.

1 Introduction

Cram is a variant of Domineering, which is a two-player perfect-information game invented by Göran Andersson around 1973. Both games were popularized to the general public in an article by Martin Gardner [7] (where Domineering was called CrossCram). Both games can be played on any subset of a square lattice, though mostly they are restricted to rectangular $m \times n$ boards, where m denotes the number of rows and n the number of columns.

Play consists of the two players alternately placing a 1×2 tile (domino) on the board. For Domineering the first player may place the tile only in a vertical alignment, the second player only horizontally; for Cram there is no such restriction: each player may place a tile vertically or horizontally. The first player being unable to move loses the game, his opponent (who made the last move) being declared the winner. Since the board is gradually filled, i.e., Domineering and Cram are converging games, the games always end, and ties are impossible. With these rules the games belong to the category of *combinatorial games*, for which a whole theory, the Combinatorial Game Theory (further CGT in short) has been developed [1,3,6].

The only difference between Domineering and Cram thus is that in the latter both players are allowed to place a domino both horizontally and vertically, i.e., in any position both players have the same possible moves. By this fact Cram belongs to the category of *impartial games* [1,3,6].

Cram has not received as much attention as Domineering in scientific research. $1 \times n$ Cram, also called Linear Cram, was proposed as early as 1934 by Dawson and completely solved by Guy and Smith in 1956 [10]. The sequence of values of the game (which is called **·07** by them) is quite irregular, but turns out to be periodic from $n = 53$ onwards, with a period of 34. Cram was further described in several sources on CGT [1,3,6], where many values were given, but mainly for small boards. For the $2 \times n$ boards it was stated [3] that all even-width boards have value $*0$ and all odd-width boards value $*1$, though no formal proof was given. The first systematic investigation of larger Cram boards was the 2009 master thesis by Martin Schneider [15]. It reported the solution of $3 \times n$ boards up to $n = 9$, and the 4×5, 5×5 and 5×7 boards. More recently Lemoine and Viennot [12] extended this to $3 \times n$ boards up to $n = 18$, $4 \times n$ boards up to $n = 9$, and $5 \times n$ boards up to $n = 8$. A few later results (3×19, 3×20, 5×9, 6×7, and 7×7) were published on-line [13].

In a previous publication [17] we reported how we have constructed CGT endgame databases for Cram for all board sizes up to 30 squares. We also provided a formal proof that $2 \times n$ boards have values $*0$ and $*1$ for even and odd n, respectively. It was further shown in a preliminary experiment that incorporating most of these databases in a simple $\alpha\beta$ solver considerably raised the solving efficiency [20]. We now have developed an elaborated $\alpha\beta$ solver for Cram aiming at solving large Cram games more efficiently and especially solving even larger games not solved hitherto. Below we report the results.

2 Combinatorial Game Theory for Cram

For combinatorial games a whole theory has recently been developed, the Combinatorial Game Theory [1,3,6]. According to this theory the main distinction within combinatorial games is in *partisan games* and *impartial games*. In partisan games both players have their own moves, for which the types and CGT values may vary wildly. Domineering belongs to this category, and a survey on different (types of) values occurring in Domineering positions was given in [18]. In CGT a game position is defined by the sets of options (moves) for the two players, conventionally named Left and Right, separately. So the notation is

$$G = \{G_1^L, G_2^L, ...|G_1^R, G_2^R, ...\},$$

where $G_1^L, G_2^L, ...$ are the games (positions) that can be reached in one move by the left player, and similarly $G_1^R, G_2^R, ...$ for the right player. Options for Left and Right are defined similarly, thus full game definitions are stated in a recursive way and can become quite complex.

For partisan games the definitions are simpler: since both players have the same moves, a game position is just defined by the set of all its options

$$G = \{G_1, G_2, ...\}.$$

If options have the same value, then all but one can be removed from the set. Moreover, for impartial games the Sprague-Grundy Theorem [9,16] states that any position is equivalent to some Nim pile and the value of that position is then always a *nimber* $*n$, where n is the size of the equivalent Nim pile.

Note that in CGT it is common to use the terms "game position" and "game value" synonymously. This stems from a powerful theorem stating that game positions with the same value act the same and are fully interchangeable (even with positions or configurations of completely different games), so are *equivalent*. Therefore, if some game position has some value x, the game position is often identified by its value and we speak just about the game x.

2.1 Nimbers

Nimbers are a special type of CGT values occuring in impartial games and sometimes in partisan games. Their main characteristic is that the options follow a certain pattern. Before we give this pattern, we first introduce some small nimbers. By reason of self-containedness we repeat a short introduction to the theory of nimbers taken from [17].

The most common nimbers occurring in Cram are the endgame $*0$ and the star game $*1$. In game notation the endgame looks as $G = \{\}$. Its value is denoted by $*0$. It denotes a terminal position where neither player has a possible move. Therefore the endgame is a loss for the player to move. Due to the equivalence of game positions with equal values it means that any non-terminal Cram position with value $*0$ is also a loss for the player to move. Besides the trivial position with no squares at all, the position consisting of a single empty square (where no domino can be placed in either direction) is the only (but frequent) terminal position with value $*0$, see Fig. 1.

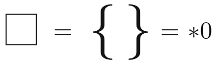


Fig. 1. A Cram endgame position.

A star is a position with just a single move, or equivalently, where each move leads to a terminal position $*0$. It looks as $G = \{*0\}$ and it is denoted by $*1$. It is of course a win for the player to move. Figure 2 shows several $*1$ positions in Cram.

Since the values of all these positions are equal, the positions are equivalent, meaning that they are interchangeable without changing the outcome of the game. The next nimbers, $*2$ and $*3$, are defined as $*2 = \{*0, *1\}$ and $*3 =$

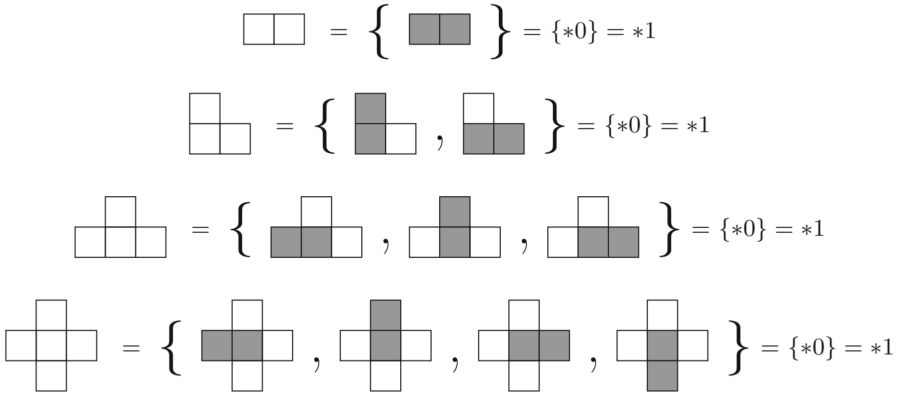


Fig. 2. Several equivalent $*1$ positions in Cram.

$\{*0, *1, *2\}$, respectively. Example $*2$ and $*3$ positions are shown in Fig. 3, where the value of the third option of the $*3$ position is $*0$, since the player to move always will lose. Note that in this and following positions symmetric options (with same values as previous options) are omitted.

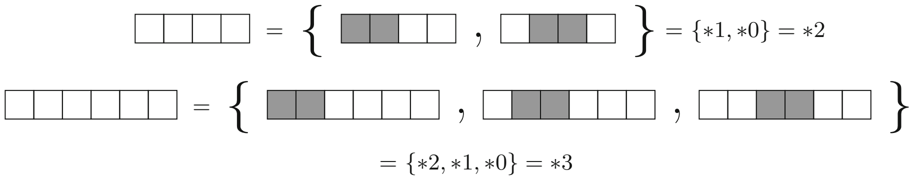
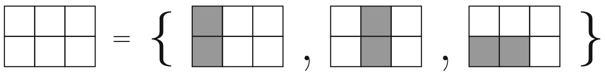


Fig. 3. Example $*2$ and $*3$ positions.

In general a nimber $*n$ is defined as $*n = \{*0, *1, ..., *(n-1)\}$. It might be tempting to think that an empty $1 \times 2n$ board has value $*n$, but that does not hold for $n > 3$. In fact the sequence of values, even for simple strips of $1 \times n$ is quite chaotic and they are difficult to obtain, except by laboriously investigating all options (recursively). Note that it is not necessarily the case that all options of a position have all values from a consecutive series from $*0$ to $*(n-1)$, to yield a $*n$ position. More precisely, the value of an arbitrary Cram position is obtained as the *mex* function applied to the values of the options of the position. The *mex* function (minimal excludant) is defined as the smallest nimber not among the options. E.g., if a position has options with values $*0$, $*1$, $*2$, and $*4$, its value is $*3$.

As an example, consider the position depicted in Fig. 4. Since all options have value $*0$ or $*2$, the position has value $*1$.



$= \{*0, *0, *2\} = *1$

Fig. 4. A $*1$ position with options with values $*0$ and $*2$ only.

This process of calculating the values of all options (and the options of the options, etc., up to endgame positions) is a laborious task, too time-consuming to execute during the search process of a solver. This is exactly why we opted for constructing Cram endgame databases filled with their nimber values once and for all.

What all nimbers $*n$ have in common is that if $n > 0$ the first player to move wins by moving to $*0$, and that if $n = *0$ the first player to move loses (all moves lead to positions with nimber value $> *0$, being wins for the next player).

2.2 Disjunctive Sums of Subgames

According to the Sprague-Grundy Theorem a Cram position with value $*n$ is equivalent to a single heap of n tokens in the game of Nim. Therefore, the theory for Nim can be applied to (sums of) Cram positions. As a consequence, if a position is decomposed into several disjoint subgames, the CGT value of the whole position is the Nim sum of the CGT values of the components. This Nim-addition rule was already discovered in 1902 by Bouton in analyzing the game of Nim [4]. In particular, if two Nim piles (or Cram fragments) have nimber values $*m$ and $*n$, their sum is given by the Nim sum $*m \oplus *n$, where $\oplus$ is the exclusive-or operator applied to the binary representations of m and n. As an example, in Fig. 5 a position is depicted with two non-interacting fragments with values $*1$ (binary representation 01) and $*2$ (binary representation 10). Since the Nim sum of 01 and 10 = 11, binary for 3, the position has value $*3$.

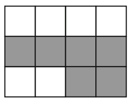


Fig. 5. A $*3$ position consisting of two fragments with values $*1$ and $*2$.

This also introduces an interesting property of nimbers: since the exclusive-or of any number with itself yields 0 ($n \oplus n = 0$) for any $n \geq 0$, every nimber is its own negative, i.e., $*n = -*n$. As a consequence, when two identical nimbers occur, they cancel each other, since their sum is equivalent to the endgame.

2.3 Cram Strategies

For empty rectangular positions in Cram it is easy to state some general observations regarding the CGT values. These were already given by Gardner [7].

Theorem 1. *Every even $\times$ even empty Cram board is a second-player win.*

Proof. The second-player's strategy to reach this is easy: after every first-player's move the second player responds by playing the symmetrical move (with respect to the centre of the board), ensuring that he/she will make the last move and win the game. Consequently, even $\times$ even empty boards have value $*0$. □

Theorem 2. *Every even $\times$ odd empty Cram board is a first-player win.*

Proof. The winning strategy of the first player is to occupy with the first move the two central squares, after which he/she plays according to the strategy given in Theorem 1 (pretending to be the second player from now on), securing the win. Consequently, even $\times$ odd (and odd $\times$ even) empty boards have values $*n$ with $n \geq 1$. □

Regarding odd $\times$ odd empty boards both theorems give no clue about their values, and no other optimal strategy is known, so they can be either first-player wins (with any positive nimber value) or second-player wins (with value $*0$).

3 Experiments and Discussion

We implemented a straightforward depth-first $\alpha\beta$ [11] solver in C# on a standard HP desktop PC with an Intel i5-4590 CPU @ 3.30 GHz. This searcher investigates lines until the end (returning that either the first or the second player made the last move and so wins the game). We further used a standard transposition table [8] based on Zobrist hashing [22] using a Deep replacement scheme [5]. This table contains 2^{26} entries of 8 bytes each, so with a total size of 0.5 GB. Further the Enhanced Transposition Cutoff (ETC) method [14] is implemented. Mirror symmetries and (for square boards) rotational symmetries are taken into account.

The CGT endgame databases used [17] consisted of all databases with a size up to 30 squares, with 1 byte per entry (representing a nimber). In total they need 3.77 GB of storage.

To make the experiments feasible, all experiments were terminated when a threshold of 1×10^9 nodes investigated is exceeded, unless noted otherwise.

3.1 Base Case

As a base case we first performed experiments with a plain $\alpha\beta$ solver without any particular heuristics[1] or CGT knowledge. We did experiments with several move-ordering heuristics, but these did not yield any significant increase in efficiency so far. This is in agreement with our experience that it is extremely difficult to foresee which moves on a board will be winning and which ones losing. Therefore we decided to run these experiments without any move ordering. This means that

[1] By a heuristic we mean a method that does not impact the final results, but only (hopefully) obtains the results faster.

in a position first all possible horizontal moves are investigated, from left to right and then down to up, and then all possible vertical moves, in the same order. Since according to Sect. 2.3 all even × even boards are losses for the first player, whereas all odd × even (and so even × odd) boards are wins for the first player, we only investigated odd × odd boards. Results using this basic $\alpha\beta$ solver are given in Table 1.

Table 1. Number of nodes investigated solving odd × odd Cram boards using several search engines (explained in the text). A '-' in an entry denotes that the board could not be solved within the threshold of 1×10^9 nodes investigated.

Board	win	BF	+ DB	+ FN	+ FS
3×3	2	7	1	1	1
3×5	1	254	1	1	1
3×7	1	7,617	1	1	1
3×9	1	415,550	1	1	1
3×11	2	14,801,591	413,522	192,004	31,109
3×13	1	–	16,061,481	3,032,649	80,917
3×15	1	–	733,255,885	22,351,558	73
3×17	2	–	–	–	627,228,056
5×5	2	73,500	1	1	1
5×7	1	49,687,795	4,496,711	4,906,301	1,978,017
5×9	1	–	–	–	142,237,539
7×7	1	–	–	–	956,663,573

In this table the first column gives the board dimensions, the second column the winner (a '1' for a first-player win and a '2' for a second-player win) and the third column (marked 'BF' for Brute Force) gives the number of nodes investigated to solve the board. The remaining columns denote the number of nodes investigated when some particular enhancement is added (discussed below). To keep the number of experiments feasible, we have chosen to always add a single new enhancement to the previous one.

3.2 Using Endgame Databases

Our next step is to investigate how the databases can increase the efficiency. For that we check during search whether the position consists of only a single or more fragment(s) available in the databases. If not, the search continues, otherwise if the position consists of a single fragment, the position is denoted as a win for the player to move (value $\geq *1$) or for the opponent (value $*0$), and the branch is pruned. If the position consists of more fragments the Nim sum of the fragments is calculated and used to value the position accordingly. The results using this database support are tabulated in Table 1 in the column marked '+ DB'.

It is clear that database support has a profound impact on the solving efficiency of our Cram solver. Of course all boards with a size of at most 30 squares just need one investigated node now, since the results are directly available from the databases. But also for all other solved boards we see a large increase in efficiency. Moreover, two boards that were not solvable within the threshold without database support are now solvable (3×13 and 3×15).

Once database support is incorporated, two move-ordering heuristics come to mind, both aiming at early splitting the position in multiple fragments.

Fragment Narrowing. The easiest of the two to implement is denoted as the *Fragment Narrowing Heuristic* (FN). According to this heuristic, for non-square boards if the board is "wide" (more columns than rows) vertical moves are always preferred over horizontal moves, and vice versa for "high" boards (more rows than columns). This is justified since moves across the short dimension will in general split the position earlier into subfragments. As a second criterion, both for vertical and horizontal moves, we prefer moves closer to the centre. The reason for this is that when a move splits the position into fragments, it is better that they are of roughly equal size than one small and one large fragment, since in the latter case the probability is higher that the larger fragment still will not be available in the databases. The results using this heuristic additionally to the previous version are tabulated in Table 1 in the column marked '+ FN'.

It is clear from the table that this heuristic also has a significant positive influence on the solving efficiency.

Fragment Splitting. As second move-ordering heuristic we make sure that moves splitting a fragment are preferred over moves just narrowing a fragment (since the narrowing has as goal an early splitting). This is implemented as the *Fragment Splitting Heuristic* (FS) and has preference (by scaling the values) over the Fragment Narrowing Heuristic. This heuristic works by summing the squared sizes of all empty fragments and deducting these from the squared total number of empty fields on the board. In this way splitting moves are favoured and especially splitting moves where the parts are roughly equal in size. A simple example calculation of a move-ordering value will clarify this heuristic[2]: in the position on the 3×19 board after 1. j2:j3 the non-splitting move 2. i2:i3 with 1 fragment of 53 squares receives an ordering value of $53^2 - 53^2 = 0$. In fact, any non-splitting move gets a value of 0. The splitting move 2. i1:i2 with 2 fragments of 25 and 28 squares receives the ordering value $53^2 - 25^2 - 28^2 = 1400$. The optimal splitting move is 2. i1:j1 with 2 fragments of 26 and 27 squares receiving the ordering value $53^2 - 26^2 - 27^2 = 1404$. The results using this heuristic additionally to the previous version are tabulated in Table 1 in the column marked '+ FS'.

[2] We use chess-like notation for moves, where squares are indicated by their column ('a', 'b', etc, from left to right) and their row ('1', '2', etc, from bottom to top). A move consists of the two squares covered separated by a colon.

Again we observe a large improvement in solving power. Moreover, another three boards that were not solvable within the threshold without the FS Heuristic are now solved (3×17, 5×9, and 7×7).

Remarkable is the increase in solving efficiency for the 3×15 board. After the opening move h1:h2 (chosen due to the FN Heuristic) a position is reached (see Fig. 6) where all possible moves by the second player are trivially shown to be losses using the databases for $3 \times n$ with $n \leq 8$.

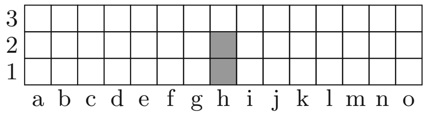


Fig. 6. The winning move h1:h2 on the 3×15 board.

Of the 66 possible moves, we only consider the 33 moves in the left half of the board (the other ones are symmetric). Using the FS Heuristic, first the moves that split the board are considered (g2:g3, g3:h3, and f3:g3). They lose immediately, since the Nim sum of the fragments after splitting is $\geq *1$. After all 30 other second moves, at level 3 first splitting moves are investigated, and for these boards it is always the case that one of them wins (leading to two fragments with Nim sum $*0$). As a consequence, the complete search tree for the 3×15 board is at most 3 deep, which leads to a tremendous reduction in tree size (from more than 22 million without the FS Heuristic to just 73 nodes with this heuristic).

3.3 Using CGT Values

In this subsection we investigate several ways to reduce the search trees further, based on using knowledge from CGT applied to Cram fragments obtained. The results are in Table 2. This table continues from the best results in Table 1 (the rightmost column marked '+ FS'), where we now omitted the results for the boards directly available from the databases. The remaining columns contain the experimental results for five enhancements explained below the table, again in an additive manner.

Skipping Moves. When the position is split in multiple fragments and one or more of these have known values (from the databases), but the value of at least one fragment is still unknown (so the search should continue), we often can omit moves in the known fragments from further consideration. We consider the following three steps:

1. In each fragment with value $*0$ all moves can be skipped from further investigation, since it denotes a losing fragment and it is never useful to play within

Table 2. Number of nodes investigated solving odd × odd Cram boards using several search engines (explained in the text). A '-' in an entry denotes that the board could not be solved within the threshold of 1×10^9 nodes investigated. In the final version (rightmost column) no threshold was set.

Board	Win	Table 1	+ SM	+ SF	+ LBS	+ SLR	+ BD
3 × 11	2	31,109	31,008	30,670	30,323	29,882	26,437
3 × 13	1	80,917	80,312	79,573	79,080	77,300	66,079
3 × 15	1	73	73	73	73	73	73
3 × 17	2	627,228,056	377,565,993	258,704,721	256,011,532	250,196,286	204,860,712
3 × 19	2	–	–	–	–	–	18,327,715,035
3 × 21	1	–	–	–	–	–	962,810,282
5 × 7	1	1,978,017	1,935,662	1,888,485	1,861,601	1,803,369	1,354,483
5 × 9	1	142,237,539	116,023,958	98,784,937	97,365,853	94,970,220	75,190,993
5 × 11	2	–	–	–	–	–	204,521,419,098
7 × 7	1	956,663,573	710,643,974	607,605,602	590,066,547	567,193,292	433,712,107

a losing fragment (every move leads to a new fragment with some value $*n$ with $n \geq 1$, to which the opponent always can just respond with a move leading to a fragment with value $*0$ again).

2. Next we check if in the remaining known fragments pairs of fragments occur with the same value. Each such pair can be omitted from further consideration, since an arbitrary move by the player to move in one fragment of such pair, changing its value to some new value, can always be responded by the opponent in playing an equivalent move (to the same new value) in the other fragment of the pair. Therefore all moves in each such pair can be skipped. This of course also follows from the fact that the Nim sum of such a pair has CGT value $*0$ ($*n + *n = *0$ for arbitrary n).
3. Finally, we calculate the sum-value of all remaining known fragments that are not skipped. If this sum-value is $*0$ the fragments together are losing and all moves in them can be skipped (e.g., consider three fragments with values $*1$, $*2$, and $*3$, with Nim sum $*0$). If the sum-value is not $*0$, but lower than or equal to the value of one of the non-skipped fragments, then all fragments except this one can be skipped, since any value that can be reached by playing in one of all non-skipped fragments, can also be reached by playing in that particular fragment.

These three ways of skipping moves based on CGT-values of fragments are taken together in the *Skip Moves Heuristic* (SM). The results using this heuristic additionally to the previous version are tabulated in Table 2 in the column marked '+ SM'.

Although the increase of efficiency is negligible for the smaller boards, it still is significant for larger boards.

Simplifying Fragments. For all remaining known (unskipped) fragments it is useful to simplify such fragments if possible. This is based on the idea that two different fragments with the same value are equivalent. So if the fragment under consideration is large but is equivalent with a much smaller fragment, we simplify it. Effectively this is done by only allowing some specific moves in the fragment. More explicity, if a fragment has value $*n$, then we only allow n moves in this fragment, one resulting in a value $*0$, one in $*1$, up to one in $*(n-1)$. All other moves in the fragment are skipped. This method is called the *Simplify Fragments Heuristic* (SF). The results using this heuristic additionally to the previous version are tabulated in Table 2 in the column marked '+ SF'.

We observe again that the increase of efficiency is negligible for the smaller boards, but significant for larger boards.

Linear and Bended Fragments. Occasionally we encountered during search a linear chain that is longer than the dimension of any of the databases used. Since Linear Cram is completely solved for arbitrary length [10], we just obtain its value by a look-up in a prefilled array. As an example, the linear chain in Fig. 7(left) is not in any of our databases, still its value is determined to be $*5$. Consequently the whole position in this figure has value $*5$ and thus the position is a win for the player to move. More often, we encounter bended chains (also called *snakes* [3,21]) that have the same value as a linear chain of the same size. So these also can be looked-up in our preprogrammed array. As an example, the snake in Fig. 7(right) has the same value $*5$ as the chain in Fig. 7(left). The recognition of linear or bended snakes is simple: the fragment must consist of squares where exactly two have one neighbour (the endpoints of the snake), and all others have exactly two neighbours.

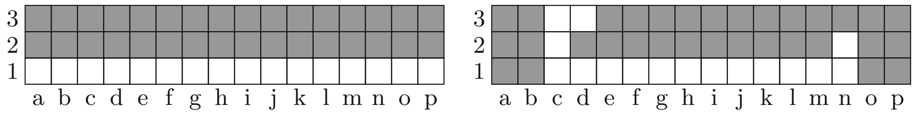

3
2
1
a b c d e f g h i j k l m n o p
3
2
1
a b c d e f g h i j k l m n o p

Fig. 7. An example of a linear chain of size 16 and a bended snake with the same size on the 3×16 board.

The method of recognizing and valuing snakes is called the *Linear/Bended Snakes Heuristic* (LBS). The results using this heuristic additionally to the previous version are tabulated in Table 2 in the column marked '+ LBS'.

Using this heuristic we observe a small increase in efficiency for all boards investigated.

Skipping Lost Responses. When in a position we investigate a move x for a player and if x turns out to be a loss after some response y by the opponent, then it is not needed for the player to investigate y itself (if not already done), since then y will lose to x now being played by the opponent. Of course this stems from the fact that the resulting position is the same (losing, so with value $*0$), and does not depend on the order of the moves played leading to the position. We denoted this as the *Skip Lost Responses Heuristic* (SLR). The results using this heuristic additionally to the previous version are tabulated in Table 2 in the column marked '+ SLR'.

Using this heuristic we observe a small but significant increase in efficiency for all boards investigated.

Bridge Destroying. In the last heuristic applied we use proofs in *Winning Ways* [3], showing that a Cram fragment with some particular pattern may be split into two fragments, such that the value of the original fragment (when not obtained from the databases) is equal to the sum of the values of the two smaller fragments (each with possibly known value). We distinguish four cases, according to the sketches depicted in Fig. 8.

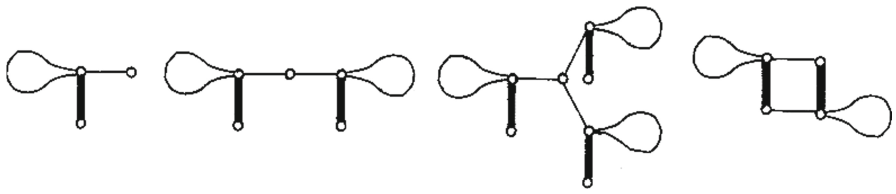


Fig. 8. Sketches of empty Cram fragments that can be broken in subfragments without changing the value. [Taken from [3], Vol. 3, p. 504.]

In these figures the dots denote empty squares, the lines denote connected (neighbouring) squares and the balloons denote arbitrary empty fragments (but not connected to the other parts of the fragment). Thin lines denote connections that may be broken without changing the value of the fragment.

The first 3 fragments have in common that there is a denoted empty square (the "bridge") connected to 1 to 3 subfragments, where each subfragment is connected to one single empty square (the bold connection) and further arbitrarily to some remainder (the balloon). Since in Cram a square has maximally four neighbours, this in principle leads to 4 such cases, but the fourth can "physically" only occur on boards where all balloons in the four subfragments are empty, thus consisting necessarily of a single empty square with four neighbouring fragments of exactly two empty squares each (a "cross"), with known value from the 5×5 database of $*0$. The first two (with one or two subfragments) occur frequently in Cram positions, the third one (with three subfragments) can physically only occur on boards with both dimensions ≥ 5 and moreover at least 1 of the three balloons empty, but still is useful.

The fourth fragment in Fig. 8 consists of four empty squares in a 2×2 arrangement with two arbitrary subfragments connected to two **opposing** squares of the 2×2 array. Note that in this case breaking the thin connections means that the fragment can be split either vertically through the 2×2 part (as depicted) or horizontally (rotated), but not diagonally. Again, the value of the whole fragment is equal to the sum of the values of the parts after splitting.

We implemented all four cases in Fig. 8 and denoted them together as the *Bridge Destroying Heuristic* (BD). The results using this heuristic additionally to the previous version are tabulated in Table 2 in the column marked '+ BD'.

Using this heuristic we observe a modest increase in efficiency for all boards investigated. We also solved a new board within the set threshold using this version of our solver, namely the 3×21 board. This board was hitherto not solved, so is our first extension of known solved Cram boards. It is a first-player win.

Since this is (for the time being) the last version of our Cram solver, we decided to also investigate a few more boards without using the 1×10^9 nodes threshold. First of all we investigated the 3×19 board (which remarkably needs much longer to be solved than the 3×21 board), requiring some 1.8×10^{10} nodes investigated. With considerably more effort we also solved the 5×11 board, needing some 2×10^{11} nodes. This board was also not solved up to date and is a second-player win.

4 Conclusions and Future Research

The results using the CGT endgame databases into our $\alpha\beta$ solver are good, with reductions of at least 90% in worst case for non-trivial boards, and of course 100% for the trivial boards (i.e., directly available from a database). This shows that the effect on solving power for the impartial game Cram is comparable to those for the partisan game Domineering [2] and the partisan all-small game Clobber [19]. By this result it is shown that this method of incorporating CGT into $\alpha\beta$ solvers by building CGT endgame databases is beneficial for a wide range of combinatorial games with the characteristic that game positions can split into independent subgames.

Using two move-ordering heuristics (the Fragment Narrowing Heuristic and the Fragment Splitting Heuristic) promoting early splitting of positions into independent fragments and thus enlarging the probability of early hitting the databases, results in another reduction in search tree size of a factor of 2 to 10 (and sometimes much more).[3] We can therefore conclude that endgame databases with CGT values are pivotal for the success of solving Cram boards.

The inclusion of five additional enhancements, based on domain-specific knowledge from Combinatorial Game Theory for Cram, enables a further large increase in solving power. Although modest per heuristic, in combination they reduce the search tree size for large boards with another factor of 2 to 3.

[3] The fact that these move-ordering heuristics are so successful does not contradict our statement that it is hard to find criteria indicating which moves are probably good or bad. It just leads to investigating first moves with small search trees (including hopefully "accidentally" winning moves).

All our results are in agreement with previous outcomes published in the literature. Moreover, we were so far able to solve two new Cram boards not solved before, the 3×21 board, being a first-player win, and the 5×11 board, being a second-player win.

Regarding future research a first idea is based on the observation that terminal nodes might vary widely regarding the depth in the search tree, even among splitting or narrowing moves. Therefore it might be useful to implement some form of iterative deepening into the search. We further still have several ideas based on Combinatorial Game Theory that might prune the search trees for Cram boards even further. Finally we consider implementing a Nimber Search program that calculates the exact CGT value for each position investigated instead of just determining if the position is a win or loss for the player to move. Comparing the results using Nimber Search with the results from our current $\alpha\beta$ solver it is interesting to investigate the claim by Lemoine and Viennot [12] that it might be more efficient to determine outcomes of Cram boards by computing nimbers for fragments than to determine the outcomes by developing directly the game tree for the sum of fragments.

References

1. Albert, M.H., Nowakowski, R.J., Wolfe, D.: Lessons in Play: An Introduction to Combinatorial Game Theory. A K Peters, Wellesley (2007)
2. 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)
3. Berlekamp, E.R., Conway, J.H., Guy, R.K.: Winning Ways for your Mathematical Plays. Academic Press, London (1982). 2nd edition, in four volumes: vol. 1 (2001), vols. 2, 3 (2003), vol. 4 (2004). A K Peters, Wellesley
4. Bouton, C.I.: Nim, a game with a complete mathematical theory. Ann. Math. **3**, 35–39 (1902)
5. Breuker, D.M., Uiterwijk, J.W.H.M., van den Herik, H.J.: Replacement schemes for transposition tables. ICCA J. **17**(4), 183–193 (1994)
6. Conway, J.H.: On Numbers and Games. Academic Press, London (1976)
7. Gardner, M.: Mathematical games. Sci. Am. **230**, 106–108 (1974)
8. Greenblatt, R.D., Eastlake, D.E., Crocker, S.D.: The Greenblatt chess program. In: Proceedings of the AFIPS Fall Joint Computer Conference, vol. 31, pp. 801–810 (1967)
9. Grundy, P.M.: Mathematics and games. Eureka **2**, 6–8 (1939)
10. Guy, R.K., Smith, C.A.B.: The G-values of various games. Proc. Camb. Philos. Soc. **52**(3), 514–526 (1956)
11. Knuth, D.E., Moore, R.W.: An analysis of alpha-beta pruning. Artif. Intell. **6**, 293–326 (1975)
12. Lemoine, J., Viennot, S.: Nimbers are inevitable. Theoret. Comput. Sci. **462**, 70–79 (2012)
13. Lemoine, J., Viennot, S.: Records. http://sprouts.tuxfamily.org/wiki/doku.php?id=records#cram. Accessed 13 May 2019

14. Plaat, A., Schaeffer, J., Pijls, W., de Bruin, A.: Nearly optimal minimax tree search? Technical report 94-19. University of Alberta, Department of Computing Science (1994)
15. Schneider, M.: Das Spiel Juvavum. Master thesis, Universität Salzburg (2009)
16. Sprague, R.P.: Über mathematische Kampfspiele. Tohoku Math. J. **41**, 438–444 (1935)
17. Uiterwijk, J.W.H.M.: Construction and investigation of Cram endgame databases. ICGA J. **40**(4), 425–437 (2018)
18. Uiterwijk, J.W.H.M., Barton, M.: New results for Domineering from combinatorial game theory endgame databases. Theoret. Comput. Sci. **592**, 72–86 (2015)
19. Uiterwijk, J.W.H.M., Griebel, J.: Combining combinatorial game theory with an α-β solver for Clobber: theory and experiments. In: Bosse, T., Bredeweg, B. (eds.) BNAIC 2016 – Artificial Intelligence. Communications in Computer and Information Science, vol. 765, pp. 78–92 (2017)
20. Uiterwijk, J.W.H.M., Kroes, L.: Combining combinatorial game theory with an alpha-beta solver for Cram. In: Atzmueller, M., Duivesteijn, W. (eds.) BNAIC 2018: 30th Benelux Conference on Artificial Intelligence, pp. 267–280. Jheronimus Academy of Data Science, 's-Hertogenbosch (2018)
21. Wolfe, D.: Snakes in Domineering games. Theoret. Comput. Sci. (Math Games) **119**, 323–329 (1993)
22. Zobrist, A.L.: A new hashing method with application for game playing. Technical report #88, Computer Science Department, The University of Wisconsin, Madison (1970). Reprinted in ICCA J. **13**(2), 69–73 (1990)