DOMINEERING: SOLVING LARGE COMBINATORIAL SEARCH SPACES

Nathan Bullock[1]

Edmonton, Alberta, Canada

ABSTRACT

Domineering, also known as crosscram, is a perfect-information, two-player game. We have created a search program which is able to prove who wins on many different sizes of boards. Some of the board sizes no one else has ever been able to solve. The main improvement is our evaluation function which can determine statically a winner at a shallower point in the search tree than was previously possible by other evaluation functions. It allows us to eliminate large portions of the search space. Along with a few other improvements, the evaluation function enabled us to solve board positions with just a fraction of the number of nodes which previous solvers needed.

1. THE GAME OF DOMINEERING

Domineering, also known as crosscram, is a perfect-information, two-player game. It is played on a board that is a subset of a square lattice. The game is played by two players who take turns placing 2×1 tiles upon the board. One player is only allowed to place a tile in a horizontal orientation and the other is only allowed to place a tile in a vertical orientation. Tiles are not allowed to overlap. The game ends when one player, the loser, is unable to place any more tiles onto the board.

Domineering was introduced by Göran Andersson around 1973 (Gardner, 1974). Since then it has been investigated from both a mathematical and an artificial-intelligence point of view.

Mathematicians have examined domineering for many years using combinatorial game theory (Berlekamp, 1988). This approach allows them to determine the combinatorial game value of a board position by taking the sum of the independent smaller positions in that position. For example if the value of the 2×2 board is known, then the value of two 2×2 game boards played together can be determined. These results have appeared in a number of books on combinatorial game theory (Berlekamp, Conway, and Guy, 1982; Conway, 1986).

Artificial-intelligence researchers have also taken an active role in looking at the game of domineering. Through the use of state-of-the-art search engines the game theoretic value of many different sizes of domineering boards have been determined (Breuker, Uiterwijk, and van den Herik, 2000). In domineering there exist four possible values for each position.

- A vertical win, denoted by V, meaning that regardless of who goes first vertical will always win.
- A horizontal win, denoted by H, meaning that regardless of who goes first horizontal will always win.
- A first-player win, denoted by $1st$, meaning that regardless of who goes first the first player will always win.
- A second-player win, denoted by $2nd$, meaning that regardless of who goes first the second player will always win.

Figure 1 shows an example of a game on a 4×5 board with vertical moving first. To refer to a move, we specify each move by the row and column number of its upper (for vertical) or left square (for horizontal) coordinates.

[1]Department of Computing Science, University of Alberta, Canada. Email:bullock@cs.ualberta.ca

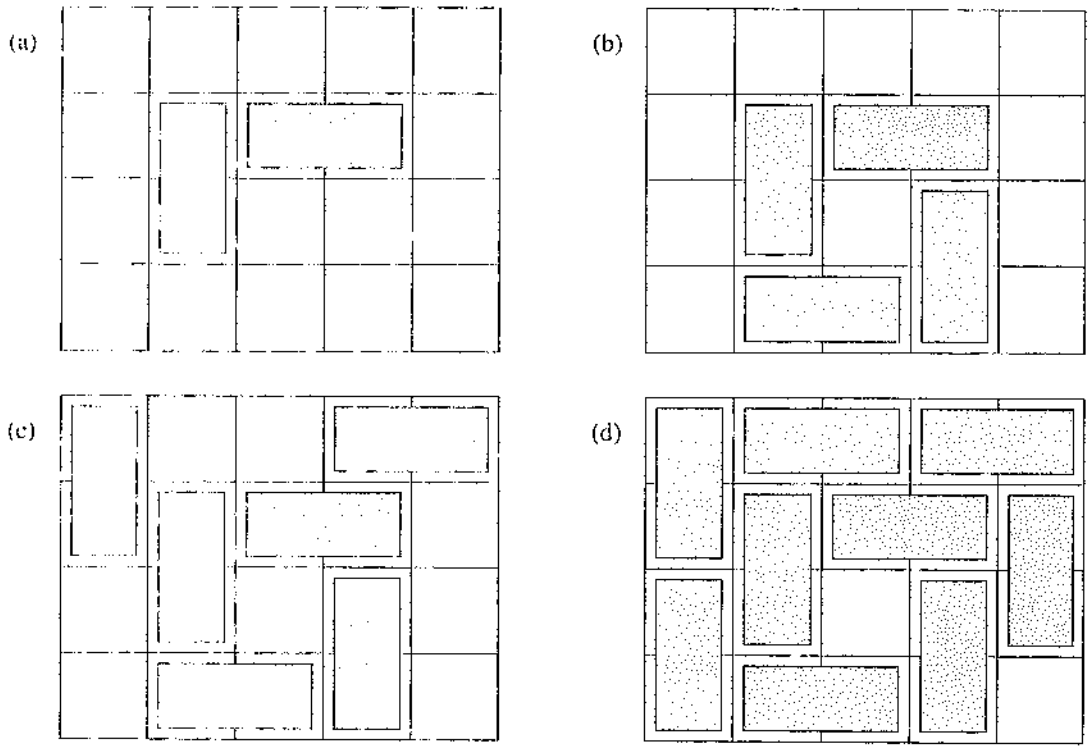

Figure 1: Game on a 4×5 board with Vertical moving first. (a) First two moves are (2,2) and (2,3). (b) Next two moves are (3,4) and (4,2). (c) Next two moves are (1,1) and (1,4). (d) Last moves are (3,1), (1,2), and (2,5). Horizontal cannot move; Vertical wins.

We label the rows from top to bottom, from 1 to the number of rows on the board, and the columns from left to right, from 1 to the number of columns on the board.

1.1 Previous Work

The first published program to solve domineering positions was the program DOMI authored by Breuker *et al.* (2000). The major strengths of DOMI are its use of transposition tables and its move-ordering heuristic. These were instrumental in enabling it to solve domineering board positions as efficiently as it did.

In 1998 when DOMI's results were gathered it was able to calculate the game theoretic values of the $m \times n$ boards where $2 \leq m \leq 8$ and $m \leq n \leq 9$. Since then, DOMI has been improved to the point where it can now solve 8×9 domineering by building a search tree approximately 3 billion nodes in size and 9×9 domineering with 25 billion nodes (Uiterwijk, 2001).

Using search techniques for solving domineering boards obviously limits the size of the largest board that can be solved. Lachmann, Moore, and Rapaport (2000) came up with a number of different rules which enable extending the results on smaller boards to determine the game theoretic values of larger boards. Through the use of these rules, Breuker *et al.*'s (2000) results, and a theoretical result for the 2×31 board, Lachmann *et al.* (2000) were able to determine the winner for all boards with 2, 3, 5, and 7 rows. Also they were able to determine the winners for all boards with 4, 9, and 11 rows, except for a finite number of smaller boards.

For a more in-depth survey of past work in domineering as well as other two-player, perfect-information games see van den Herik, Uiterwijk, and van Rijswijck (2002).

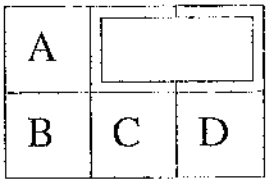

Figure 2: An example of a 2×3 domineering board where horizontal has placed a tile on position (1,2).

1.2 Overview

In this article we show that one effective way to reduce the size of the proof trees, for different sizes of domineering boards, is to come up with a more advanced evaluation function. This improved evaluation function is able to determine the winner of a game at a shallower point in the search tree, allowing us to prune a potentially large number of positions from the search.

This substantial pruning of positions from the search tree has enabled us to solve the standard domineering board, 8×8, in a matter of seconds where previously it had taken hours. It has also enabled our domineering solver, OBSEQUI which we describe in this article, to determine the game theoretic values for a few previously unsolved game boards, notably 4×19, 4×21, 6×14, 8×10, and 10×10, which has filled in a number of the holes in Lachmann *et al.*'s (2000) original results. Figure 16 (see the Appendix) provides an updated chart of those boards for which we calculated the game theoretic value.

The source code for OBSEQUI is publicly available at www.cs.ualberta.ca/~games/domineering.

2. DETERMINING WHO WINS

The number of nodes in a typical search tree grows exponentially with the depth of the tree. Given this fact it is easy to see that it is desirable to determine the winner in a game at the shallowest point possible, since this prevents us from searching a large portion of the tree.

In OBSEQUI we determine a lower bound on the number of moves that a player, α, could make, given the current board position and using a certain strategy. Then we determine an upper bound on the number of moves that α's opponent, β, could make, given the strategy that α used to get their lower bound. If α's lower bound is greater than (or equal to, depending on who just moved) β's upper bound then we can conclude that α wins. Similarly we can also do this for β and determine if they win. In this section we let α denote the player for whom we are trying to determine a lower bound, or in other words the player for whom we are trying to determine if they win given a specific board position. We refer to α's opposition as β.

2.1 Definitions

We define an *unoccupied* square as one which is not currently covered by a tile. Conversely an *occupied* square is one which is covered by a tile or which is not part of the board.

An *available* square is a square which a player can place one of their tiles across. A square is *unavailable* for a player if its borders are occupied in such a way that there is no way for that player to place a tile on that square. In Figure 2 the square marked by an A is unavailable for the horizontal player while the squares B, C, and D are all available.

We can also look at squares from the perspective of where a player's opponent can play. A *protected* square is one which the opponent is unable to play on; from the opponent's perspective this square would be considered unavailable. An *unprotected* square is one which the opponent could place a tile upon; the opponent would consider this square available. In Figure 2 the square marked by A is protected for the vertical player, while the squares B, C, and D are all unprotected for the vertical player.

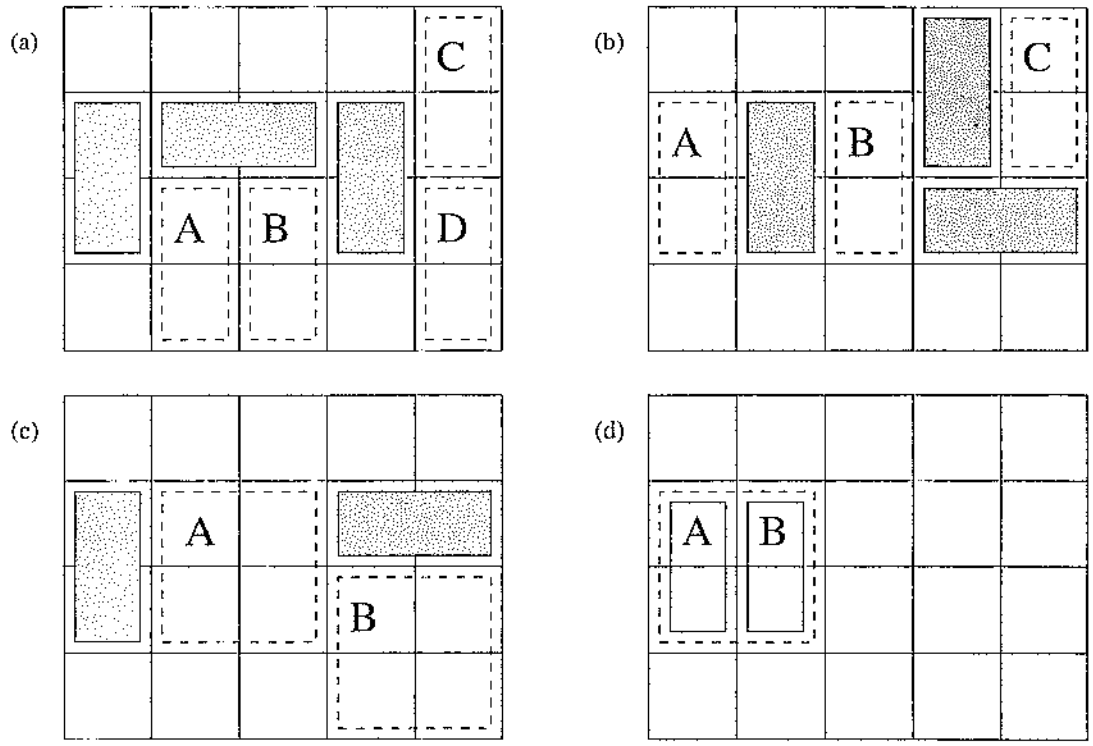

Figure 3: Board properties from the perspective of the vertical player. (a) A, B, C, and D are vulnerable areas. (b) A, B, and C are safe areas. (c) A and B are both protective areas. (d) The square outlined with a dotted line is a protective area, A is where a safe area would be created if Vertical played a tile on position B.

2.2 Board Properties

In order to determine a lower bound on the number of moves a player can make, we define three types of board properties which we use to calculate this bound. These three properties are: *vulnerable* areas, *safe* areas, and *protective* areas. Note that these properties are determined separately for the horizontal and vertical player.

A *vulnerable area* for a player, α, is a pair of adjacent squares where α can place a tile (see Figure 3(a)).

A *safe area* is a vulnerable area where both squares are protected squares for α, or in other words unavailable to β (see Figure 3(b)).[2] It is easy to see that it is impossible for β to place a tile which would overlap with one of α's safe areas. The concept of a safe area (or safe move) was also used in Breuker *et al.* (2000).

A *protective area* for a player, α, is a 2×2 unoccupied region of the board where one of the sides is bordered by occupied squares in such a way that α is able to place a tile, completely inside this area, in such a way that the other 2 squares, not covered by α's tile, form a safe area for α afterwards (see Figures 3(c) and 3(d)).

Important: Two of α's areas are considered adjacent if a single one of β's tiles could overlap both areas.

2.3 Board Cover

In order to use these board properties to get bounds on the number of moves a player has remaining, we need to determine how many of these properties exist on a given board. The rules for covering the board with these various properties are:

1. No two areas can overlap. In other words, no square can be contained within two different areas.

[2]Note that the terms safe and vulnerable may be a poor choice of words since a safe area is a subset of a vulnerable area. Nevertheless, we still feel that the terms accurately describe the properties of the areas.

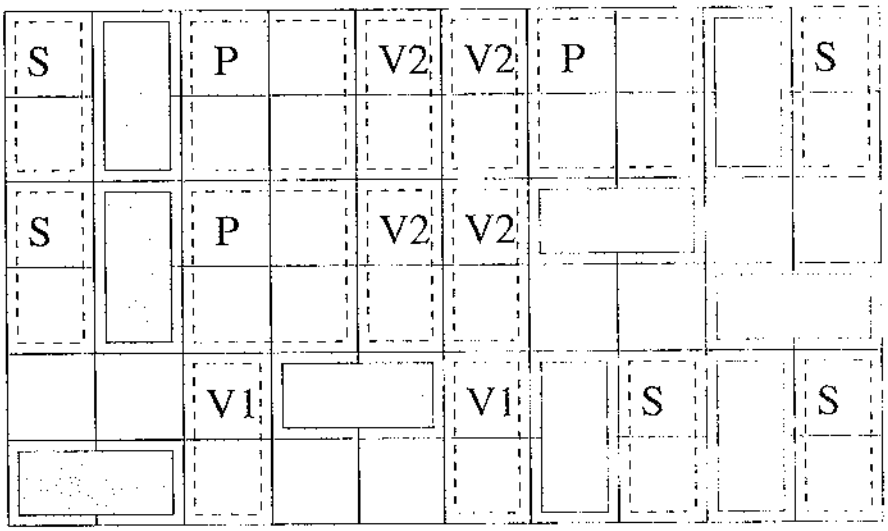

Figure 4: A possible covering of a 6×10 board for the vertical player. P denotes a protective area, $V2$ a type-2 vulnerable area, $V1$ a type-1 vulnerable area, and S a safe area.

2. No two protective areas can be adjacent. For example, the covering of the board in Figure 3(c) would be invalid for the vertical player since areas A and B are adjacent.

To improve the lower bound on the number of moves a player has left, given a certain board position, we distinguish two types of vulnerable areas: those which are not adjacent to any board property are type-1 vulnerable areas and those which are adjacent to other board properties are type-2 vulnerable areas. This naming scheme is to make it clear that β, with a single tile, could overlap up to two of α's type-2 vulnerable areas, but only one of α's type-1 vulnerable areas. For example, A and B in Figure 3(a) are type-2 vulnerable areas, while C and D are type-1 vulnerable areas.

An example of a complete covering of a board is shown in Figure 4.

2.4 Game-Playing Strategy

Given a certain board position, we want to determine a strategy for α which allows us to determine a lower bound on the number of moves α can make.

The strategy is fairly simple. First we assume that it is currently β's turn. We will show later that this assumption does not really affect the size of the proof tree. Then we reply to each move β makes in the following way.

- If β places a tile over at least one of α's areas, then α responds by playing in the same type of area as β's tile overlapped. More explicitly:
  1. If β places a tile over one of α's protective areas, and this was not α's last protective area, α will respond by playing in another protective area. Note that this applies even if β's tile overlapped both a protective and a type-2 vulnerable area.
  2. Else, if β places a tile over one or two of α's type-2 vulnerable areas, and these were not α's last type-2 vulnerable areas, α responds by playing in another of these areas.
  3. Else, if β places a tile over one of α's type-1 vulnerable areas, and this was not α's last type-1 vulnerable area, α responds by playing in another of these areas.
  4. Else, if there are no areas remaining of the same type as that which β placed their tile upon, α responds by playing in any available area type.
- If β's tile does not overlap any of α's area types, then α can play a tile in either a safe area, a vulnerable area, or a protective area.

By following this strategy α is guaranteed to get some fraction of each type of area which they have marked in their board cover. Furthermore, by looking at the properties of the board and the number of moves α was able to play, we can determine an upper bound on the number of moves β can make. Finally, by comparing these upper and lower bounds, and taking into account whose turn it is, we can determine if α will win by following this game-playing strategy.

We have no illusions that this strategy is an optimal one, but from the results shown in Subsection 5.1 it appears to be very effective for computing bounds. What makes our strategy nice is that it is simple, makes sense intuitively, and is formulated in a way which makes it easy to prove bounds on the number of moves each player can still make. The main goal of this approach is to recognize easy wins and losses early in our search.

2.5 Lower Bound on Number of Moves Remaining

Now that we have defined a set of board properties, rules for creating a board cover, and a game strategy, we can compute a reasonable lower bound on the number of moves a player has left. We refer to the number of protective areas that exist in α's covering of the board as $prot(\alpha)$, the number of type-2 vulnerable areas as $vuln2(\alpha)$, the number of type-1 vulnerable areas as $vuln1(\alpha)$, and the number of safe areas as $safe(\alpha)$.

Theorem 1 *If it is currently β's turn, and α plays with the strategy given in Subsection 2.4, then α has at least $moves(\alpha)$ left where:*

$$moves(\alpha) = 2 \cdot \left\lfloor \frac{prot(\alpha)}{2} \right\rfloor + \left\lfloor \frac{vuln2(\alpha)}{3} \right\rfloor + \left\lfloor \frac{vuln1(\alpha)}{2} \right\rfloor + safe(\alpha) + f(\alpha)$$

Where

$$f(\alpha) = \begin{cases} 1 & \text{if } vuln2(\alpha) \bmod 3 \neq 0 \text{ and } vuln1(\alpha) \bmod 2 \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

A few special cases can be removed in the case where $prot(\alpha) \bmod 2 = 1$, by canceling one of the protective areas in α's cover of the board and converting it into two type-2 vulnerable areas. This guarantees that $prot(\alpha) \bmod 2 = 0$, simplifying the above equation and the proof.

Proof: We know from our general game strategy, given in Subsection 2.4, that α attempts to reply to any of β's moves by playing in the same type of area which β's tile overlapped. From this we can guarantee α will be able to place their tiles in some fraction of each type of area which has been included in the board cover.

1. Due to the nature of a safe area, namely the opponent cannot place a tile which will overlap it, α will be able to play a tile in every one of the safe areas which are marked on the board regardless of how β plays. Thus α is guaranteed $safe(\alpha)$ moves.

2. Since β can only block one of α's protective areas with each of their tiles and then the very next turn α will place a tile within one of these protective areas, α will be able to occupy at least $\left\lfloor \frac{prot(\alpha)}{2} \right\rfloor$ of those areas. Also we know that for each protective area that α places a tile in, α also creates another safe area for themselves. Therefore each protective area which α is able to occupy gives them 2 moves. Hence, α's protective areas guarantee α at least $2 \cdot \left\lfloor \frac{prot(\alpha)}{2} \right\rfloor$ moves.

   It is possible for β to overlap both a protective area and a type-2 vulnerable area with one tile (see Figure 5(a)). However in doing this β covers only one square in the protective area. Therefore, from the remaining portion of α's protective area, another type-2 vulnerable area can be created to make up for the one which was lost (see Figure 5(b)).

3. Since β can place a tile over at most two of α's type-2 vulnerable areas with each move and α can respond by placing a tile in another type-2 vulnerable area, α will be able to occupy at least $\left\lfloor \frac{vuln2(\alpha)}{3} \right\rfloor$ of their type-2 vulnerable areas.

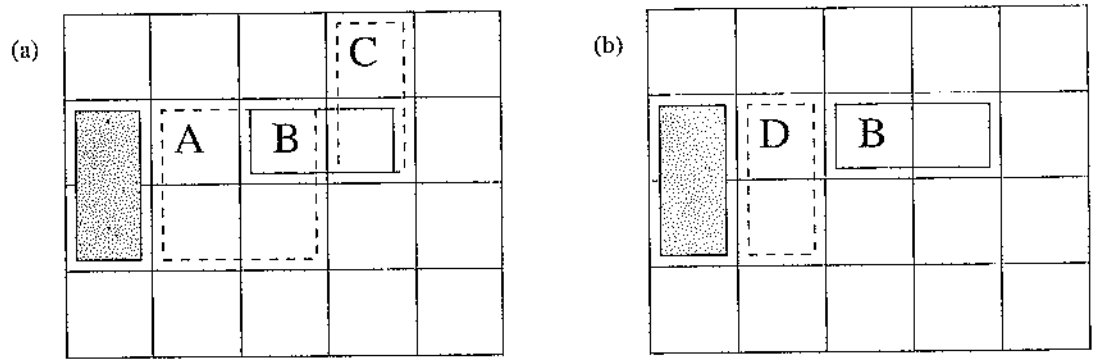

Figure 5: Crossing a protective and type-2 vulnerable area. (a) Opponent places a tile, B, across both a protective area, A, and type-2 vulnerable area, C. (b) Player cancels both A and C and creates a new type-2 vulnerable area, D.

4. Similarly, since β can place a tile over at most one of α's type-1 vulnerable areas with each move and α can respond by placing a tile within another type-1 vulnerable area, α will be able to occupy at least $\left\lfloor \frac{vuln1(\alpha)}{2} \right\rfloor$ of their type-1 vulnerable areas.

There is one special case that we should take note of. If $vuln2(\alpha) \bmod 3 \neq 0$ and $vuln1(\alpha) \bmod 2 \neq 0$ then there will be a point when β will have to place a tile across either one or two remaining type-2 vulnerable areas or β will have to place a tile over a remaining type-1 vulnerable area. In either case α will not be able to respond by playing on that same type of area, but will have to respond by placing a tile on the other type of vulnerable area. Since the number of this other type of area is not evenly divisible by either 3 or 2 respectively, α will be guaranteed at least one more move of this type. □

A lower bound on the number of moves that α has remaining if it is currently α's turn to move can also be easily calculated. We can simply place a tile on the board, according to some heuristic, and then calculate the lower bound after that move.

2.5.1 Example of Lower Bound

Consider the board covering given in Figure 4, let α be the vertical player, and assume that it is Horizontal's turn to play. We can use Theorem 1 to determine a lower bound on the number of moves that Vertical can make.

In Figure 4, $prot(\alpha) = 3$, $vuln2(\alpha) = 4$, $vuln1(\alpha) = 2$, and $safe(\alpha) = 5$. Since $prot(\alpha) \bmod 2 = 1$ we convert one protective area into two type-2 vulnerable areas, leading to $prot(\alpha) = 2$, $vuln2(\alpha) = 6$, $vuln1(\alpha) = 2$, and $safe(\alpha) = 5$. Now we can use our formula:

$$\begin{aligned} moves(\alpha) &= 2 \cdot \left\lfloor \frac{prot(\alpha)}{2} \right\rfloor + \left\lfloor \frac{vuln2(\alpha)}{3} \right\rfloor + \left\lfloor \frac{vuln1(\alpha)}{2} \right\rfloor + safe(\alpha) + f(\alpha) \\ &= 2 \cdot \left\lfloor \frac{(2)}{2} \right\rfloor + \left\lfloor \frac{(6)}{3} \right\rfloor + \left\lfloor \frac{(2)}{2} \right\rfloor + (5) + (0) \\ &= 10 \end{aligned}$$

No matter how β plays, α can play at least 10 more tiles. Note that Breuker *et al.* (2000) would obtain a lower bound of 5.

2.6 Upper Bound on Number of Moves Remaining

We now would like to place an upper bound on the number of moves that β could play, given a specific board position and assuming that α will play according to the strategy given in Subsection 2.4.

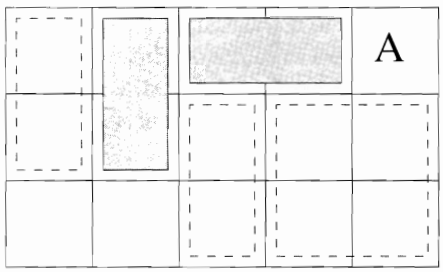

Figure 6: The square, A, is not covered by Vertical's board covering and is not available to Horizontal. (The dotted rectangles denote Vertical's board covering.)

In the previous subsection we determined that α is guaranteed, regardless of how β plays, to be able to play at least $moves(\alpha)$ more tiles onto the board. Therefore, this enables us to look at what properties the game board will have after α has placed all of these tiles, realizing of course that α must follow the strategy given in Subsection 2.4 and that we do not know how β will play.

Since α has at least $moves(\alpha)$ turns, β has at least $2 \cdot moves(\alpha)$ fewer unoccupied squares available to play their tiles on. Let $squares(\beta)$ denote the number of squares which are still unoccupied after α has placed their tiles upon the board.

Further there can be squares on the board which are not covered by α's board cover, and are yet unavailable to β. Let $unavail(\beta)$ denote the number of squares which fit this description (see Figure 6). These squares will never be played on since β is unable to and α's strategy does not take them into consideration.

Finally, there are a number of squares which are included in α's covering of the board but which, due to the properties of these squares and of what we know of how α will play their tiles, α will not actually cover with a tile and β will be unable to play on. Let $unplayable(\beta)$ denote the number of these types of squares. We describe them in more detail below.

2.6.1 Unplayable Squares

To determine the number of squares which are included in α's board cover but which neither α will cover with a tile nor which β will be able to play on, we define two more types of board properties.

An *option area* for α is an unprotected, unoccupied square which is appended to an already existing safe area, in such a way as to create a 1×3 rectangle (see Figure 7(a)). To include an option area in a board cover, it cannot be adjacent to any other board properties contained in the board cover (see Figure 7(b)).

There are three types of option areas, type 1, 2, and 3, corresponding to the number of squares which they make unavailable for β if the option is played (see Figure 7(c)). We note that the set of type-3 options is a subset of the set of type-2 option since it makes at least 2 squares unavailable to β. Similarly the set of type-2 options is a subset of the set of type-1 option. We refer to the number of each of these different areas as $op1(\alpha)$, $op2(\alpha)$, and $op3(\alpha)$ respectively.

A *vulnerable area with a protected square* for α is a type-1 or type-2 vulnerable area in which one of the squares is unavailable to β. The importance of these vulnerable areas with protected squares is that for each of these types of areas which α does not place a tile within $unplayable(\beta)$ can be increased by one. We refer to the number of each of these different areas as $vuln1\_p(\alpha)$ and $vuln2\_p(\alpha)$.

Important: The squares which these areas cover are the only ones which will be counted twice in our board cover, each of these areas will be counted as both a vulnerable area and a vulnerable area with a protected square.

To account for these new properties we refine the strategy given in Subsection 2.4 a little bit. First, if α has the choice of playing in a type-1 vulnerable area which does not contain a protected square and one which does, α will choose to play in the one which does not. Same thing for the type-2 vulnerable areas. Since we are still playing in type-1 or type-2 vulnerable areas when we are supposed to, this does not affect the proof

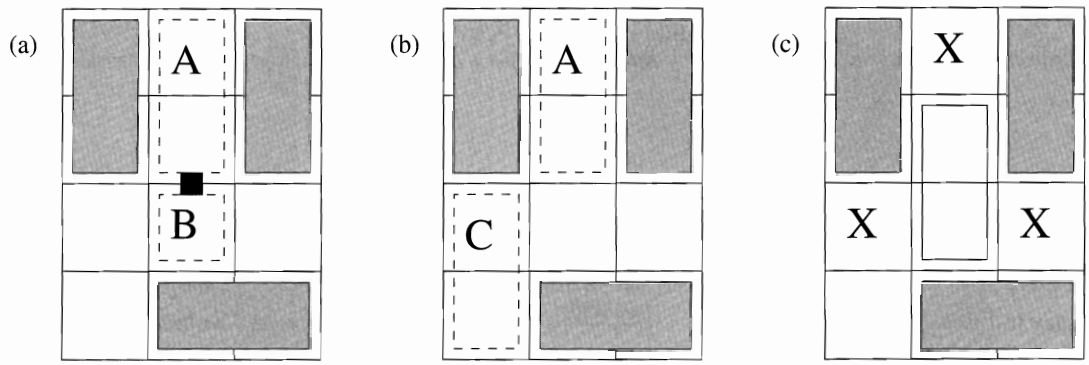

Figure 7: Option Area for vertical. (a) B marks a type 3 option area for the safe area A. (b) If C is in the board cover, B couldn't be added. (c) All of the squares denoted by an X would be unavailable for the opponent if vertical took the option.

of Theorem 1. Second, to any move made by β which overlaps one of α's option areas, α will respond by taking one of its options which is of the same value as what β covered. This also does not affect the proof of Theorem 1 since β's move would not overlap any of α's other areas.

Theorem 2 *If it is currently β's turn, and α plays with the strategy given in Subsection 2.4, then there are at least $unplayable(\beta)$ squares which are covered in α's covering of the board, which will not be used by any of α's $moves(\alpha)$ tiles and which will not be available to β where:*

$$\begin{aligned} unplayable(\beta) \;=\; & \left(vuln2\_p(\alpha) \quad \left(\left\lfloor \tfrac{vuln2(\alpha)}{3} \right\rfloor \quad \left\lfloor \tfrac{vuln2(\alpha) \;\; vuln2\_p(\alpha)}{3} \right\rfloor\right)\right) \\ & + \left(vuln1\_p(\alpha) \quad \left(\left\lfloor \tfrac{vuln1(\alpha)}{2} \right\rfloor \quad \left\lfloor \tfrac{vuln1(\alpha) \;\; vuln1\_p(\alpha)}{2} \right\rfloor\right)\right) \\ & + 3 \cdot \left\lfloor \tfrac{op3(\alpha)}{2} \right\rfloor + 2 \cdot \left\lfloor \tfrac{op2(\alpha)}{2} \right\rfloor + \left\lfloor \tfrac{op1(\alpha)}{2} \right\rfloor + f(\alpha) + g(\alpha) \end{aligned}$$

and

$$f(\alpha) = \begin{cases} 1 & \text{if } vuln2(\alpha) \bmod 3 \neq 0 \text{ and } vuln1(\alpha) \bmod 2 \neq 0 \\ & \text{and one of } vuln2\_p(\alpha) > 0 \text{ or } vuln1\_p(\alpha) > 0. \\ 0 & \text{otherwise.} \end{cases}$$

and

$$g(\alpha) = \begin{cases} 0 & \text{if } vuln2(\alpha) \bmod 3 \neq 0 \text{ and } vuln1(\alpha) \bmod 2 \neq 0. \\ 0 & \text{else if } vuln2(\alpha) \bmod 3 = 0 \text{ and } vuln1(\alpha) \bmod 2 = 0. \\ 3 & \text{else if } op3(\alpha) \bmod 2 = 1. \\ 2 & \text{else if } op2(\alpha) \bmod 2 = 1. \\ 1 & \text{else if } op1(\alpha) \bmod 2 = 1. \\ 0 & \text{otherwise.} \end{cases}$$

To remove a few special cases from the proof:

- *If $prot(\alpha) \bmod 2 = 1$, cancel one of the protective areas in α's cover of the board and convert it into two type-2 vulnerable areas. Hence, $prot(\alpha) \bmod 2 = 0$.*
- *If neither of the first two cases in $g(\alpha)$ apply, make it so that all of $op3(\alpha)$, $op2(\alpha)$, and $op1(\alpha)$ have an even value, except one, by converting option areas to smaller types as needed. For example, if you had one of each of the different types of option areas, and since a type 2 is a subset of a type 1, you could label the type-2 option area as a type 1 leaving one type-3 option area and two type-1 option areas.*
- *If one of the first two cases in $g(\alpha)$ apply, make it so that all of $op3(\alpha)$, $op2(\alpha)$, and $op1(\alpha)$ have an even value by converting option areas to smaller types as needed. For example, if you had one of each*

of the different types of option areas, and since a type 3 is a subset of a type 2, you could label the type-3 option area as a type 2 and completely remove the type-1 option area leaving two type-2 option areas.

Proof:

1. By the strategy given in Subsection 2.4, and our slight refinements given above, α will be able to place a tile in at least $\left\lfloor \frac{vuln2(\alpha) \quad vuln2\_p(\alpha)}{3} \right\rfloor$ of their type-2 vulnerable areas which do not have a protected square before they could possibly run out of these types of areas. Therefore they will not have to play in $\left(vuln2\_p(\alpha) \quad \left(\left\lfloor \frac{vuln2(\alpha)}{3} \right\rfloor \quad \left\lfloor \frac{vuln2(\alpha) \quad vuln2\_p(\alpha)}{3} \right\rfloor\right)\right)$ of their type-2 vulnerable areas which contain a protected square, each of which leaves one square which β cannot use.
2. An identical argument can be used to determine the number of type-1 vulnerable areas which contain a protected square that α will not have to place a tile within.
3. For the option areas, each time β blocks one of α's option areas, α will be able to respond immediately by placing a tile within one of their other option areas of an equal value to what β just blocked. Therefore, we can see that we will be able to use at least $\left\lfloor \frac{op(\alpha)}{2} \right\rfloor$ of each option type. We can also see that for each type-3 option we use it makes three squares unavailable for β, two for type-2 options, and one for type-1 options.

Finally, there are two special cases which need to be taken care of. The first one is just an extension of the special case in the previous proof, where $vuln2(\alpha) \bmod 3 \neq 0$ and $vuln1(\alpha) \bmod 2 \neq 0$. In Theorem 1 we were able to show that this special case enabled α to place one more tile on one of the two types of vulnerable areas. This means that unless both $vuln2\_p(\alpha)$ and $vuln1\_p(\alpha)$ equal zero, this extra move that α plays could possibly occupy one more of their vulnerable areas which contain a protected square. This would mean that we over-calculated the number of squares in α's cover which will be unavailable to β by one.

The second special case is when α has only one of $vuln2(\alpha) \bmod 3 \neq 0$ or $vuln1(\alpha) \bmod 2 \neq 0$ and an uneven number of one of their option types. This will mean that at some point in the game β will have to play across one of these two types of areas, and α will be able to respond in the other. This will give α either one more option or one more vulnerable move. If they receive the extra option then this removes up to 3 squares from β (the value of the option area). If they receive the extra vulnerable move this removes two squares from β (the area of the extra tile α can play) and β will need two more squares than they did before since α will now be able to play $moves(\alpha) + 1$ tiles. Since no options are worth more than 4 squares it is always in β's interest to count the value of the option and forget about giving α an extra move.□

2.6.2 Available Squares for Opponent

We have now determined the number of unoccupied squares β will have remaining after α has played their $moves(\alpha)$ tiles, the number of these squares which were not contained in α's board cover which are not available to β, and the number of squares within α's cover which will not be available for β. From all of this we can conclude that β can play a maximum of $\left\lfloor \frac{avail(\beta)}{2} \right\rfloor$ moves, where

$$avail(\beta) = squares(\beta) \quad unavail(\beta) \quad unplayable(\beta).$$

2.6.3 Example of Upper Bound

Consider the example given in Subsection 2.5.1 and the augmented board covering given in Figure 8. From these and Theorem 2 we determine an upper bound on the number of moves that Horizontal can make.

In Subsection 2.5.1 we already showed that α, the vertical player, can still play at least 10 tiles, with $prot(\alpha) = 2$, $vuln2(\alpha) = 6$, $vuln1(\alpha) = 2$, and $safe(\alpha) = 5$. In Figure 8 we can also see $squares(\beta) = 22$, $unavail(\beta) = 0$, $op1(\alpha) = 2$, $op2(\alpha) = 1$, $op3(\alpha) = 0$, $vuln2\_p(\alpha) = 0$, and $vuln1\_p(\alpha) = 1$.

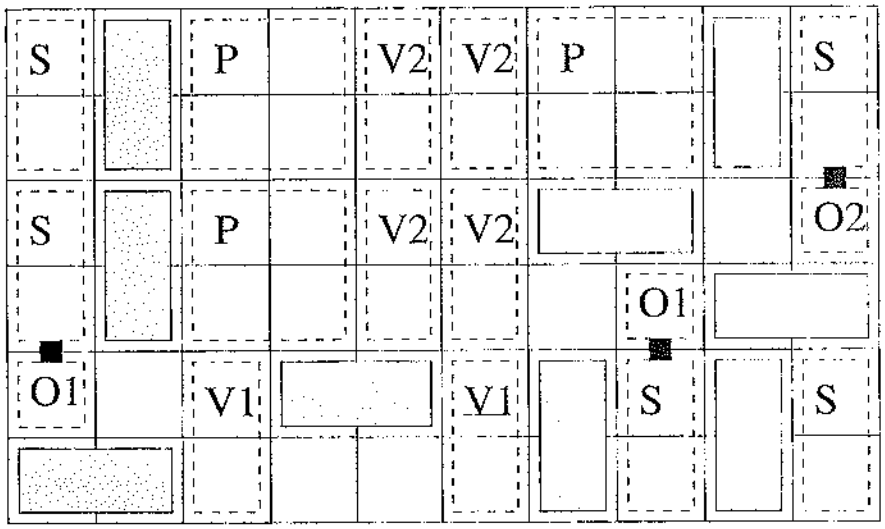

Figure 8: A possible covering of a 6×10 board for the vertical player. The squares denoted by $O1$ and $O2$ are option areas and the underlined $V1$ denotes a type-1 vulnerable area with a protected square.

Now we can use our formula:

$$\begin{aligned}
unplayable(\beta) &= \Big(vuln2_p(\alpha) \quad \Big(\Big\lfloor \tfrac{vuln2(\alpha)}{3} \Big\rfloor \quad \Big\lfloor \tfrac{vuln2(\alpha) \quad vuln2_p(\alpha)}{3} \Big\rfloor\Big)\Big) \\
&\quad + \Big(vuln1_p(\alpha) \quad \Big(\Big\lfloor \tfrac{vuln1(\alpha)}{2} \Big\rfloor \quad \Big\lfloor \tfrac{vuln1(\alpha) \quad vuln1_p(\alpha)}{2} \Big\rfloor\Big)\Big) \\
&\quad + 3 \cdot \Big\lfloor \tfrac{op3(\alpha)}{2} \Big\rfloor + 2 \cdot \Big\lfloor \tfrac{op2(\alpha)}{2} \Big\rfloor + \Big\lfloor \tfrac{op1(\alpha)}{2} \Big\rfloor + f(\alpha) + g(\alpha) \\
&= \Big((0) \quad \Big(\Big\lfloor \tfrac{(6)}{3} \Big\rfloor \quad \Big\lfloor \tfrac{(6) \quad (0)}{3} \Big\rfloor\Big)\Big) \\
&\quad + \Big((1) \quad \Big(\Big\lfloor \tfrac{(2)}{2} \Big\rfloor \quad \Big\lfloor \tfrac{(2) \quad (1)}{2} \Big\rfloor\Big)\Big) \\
&\quad + 3 \cdot \Big\lfloor \tfrac{(0)}{2} \Big\rfloor + 2 \cdot \Big\lfloor \tfrac{(1)}{2} \Big\rfloor + \Big\lfloor \tfrac{(2)}{2} \Big\rfloor + (0) + (0) \\
&= 1
\end{aligned}$$

To determine the upper bound on the number of squares β can place a tile on we use the formula from Subsection 2.6.2:

$$\begin{aligned}
avail(\beta) &= squares(\beta) \quad unavail(\beta) \quad unplayable(\beta) \\
&= (22) \quad (0) \quad (1) \\
&= 21
\end{aligned}$$

Therefore we can determine that β can play at most $\lfloor \frac{21}{2} \rfloor = 10$ tiles. Note that Breuker *et al.* (2000) would obtain an upper bound of 15.

2.7 The Winner Is ...

In a given board position α can still make at least $moves(\alpha)$ moves, no matter what α's opponent does, provided that α plays according to our given strategy. We also know that α's opponent can place at most $\lfloor \frac{avail(\beta)}{2} \rfloor$ tiles. If it is currently β's turn and $moves(\alpha) \geq opp\_moves(\alpha)$, then α can win. Similarly, if it is currently α's turn and $moves(\alpha) > opp\_moves(\alpha)$, then α can win.

From our examples in Subsection 2.5.1 and Subsection 2.6.3 we know α can play at least 10 more tiles and β can play at most 10 tiles. Therefore, since it is currently β's turn we can deduce that α has a winning move in this position. Breuker *et al.*'s (2000) bounds do not allow them to be able to solve this position statically and therefore they are forced to build a large search tree to determine the same result.

Board Size	Safe Moves	No Safe Moves
7×8	1,030,221	949,209
7×9	7,472,487	6,052,516
8×8	2,272,909	2,023,301

Figure 9: Proof-tree sizes for various boards with safe moves either being generated or not.

3. SEARCH ENHANCEMENTS

All search programs, when optimized for a specific search task, will use a number of different enhancements to the normal alpha-beta search algorithm. These enhancements could include iterative deepening, transposition tables, move-ordering heuristics, specialized local searches, and many other general as well as problem-specific ideas.

One enhancement which we investigated was proving that one type of move is always inferior to some other type of move (called: dominance). This allowed us to prune these moves from the search.

3.1 Ignoring Safe Moves

OBSEQUI is able to take advantage of the idea that as long as there exists a vulnerable area for the current player there is no need to place a tile in a safe area.

Theorem 3 *Given a domineering board which contains at least one safe area and one vulnerable area for α and it is α's turn to play. Then there exists a vulnerable area which is as good or better a move for α than to play in any of the safe areas.*

Proof: Assume that $m_1(\alpha), m_1(\beta), m_2(\alpha), m_2(\beta), \ldots$ is the optimal move sequence for α and β until the end of the game and that $m_1(\alpha)$ was placed in a safe area. There are two cases we need to look at.

Case 1. At least one of α's moves was placed in a vulnerable area. Let $m_x(\alpha)$ be the first vulnerable move α played. Since $m_x(\alpha)$ is the first vulnerable move, $m_1(\alpha), m_2(\alpha), \ldots, m_{x-1}(\alpha)$ are all played in safe areas. α achieves the same results by playing $m_x(\alpha)$ first, then $m_1(\alpha), m_2(\alpha), \ldots, m_{x-1}(\alpha)$, since $m_1(\alpha), m_2(\alpha), \ldots, m_{x-1}(\alpha)$ are all safe areas and therefore β could not interfere with them.

Case 2. None of α's moves were placed in a vulnerable area. Let $m_1(\alpha)$ be replaced with an existing vulnerable move. Since this vulnerable move can disrupt at most one of α's safe areas, and it can only decrease the amount of space β has to place their moves, it can not negatively affect α. □

3.2 Analysis

This enhancement is a very simple idea to help reduce the branching factor in the game of domineering. Figure 9 shows a number of tests which show that this enhancement does have a small effect on the size of the proof trees for various sizes of boards. Node expansion also becomes more efficient since there are fewer child positions to examine and evaluate. We will refer to these types of relationships, where one move is guaranteed to always be at least as good or better than another, as a dominance relationship.

As more dominance relations are discovered the branching factor of domineering will continue to be reduced. This could be a very promising area for further research, which would lead to further reductions in the size of the proof trees for various domineering boards.

4. SOLVING 10×10 DOMINEERING

One of the goals we wanted to reach in doing research on domineering was to solve larger boards than had ever been solved before. The pinnacle is the 10×10 board – the smallest interesting-sized unresolved problem.

Depth	Branching Factor	Nodes at Current Depth	Effective Branching Factor
0	90	1	25
1	86	25	77
2	83	1944	41
3	79	81609	43
4	77	3520539	48
5	74	169944142	-

Figure 10: Growth of 10×10 Search Tree.

Previous to our research, 8×8 domineering as well as many other smaller board sizes, had been solved by various people. The game of 9×9 domineering was reported to have been solved by DOMI in Van den Herik, Uiterwijk, and Van Rijswijck (2002). We wanted to try the next step: 10×10 domineering.

4.1 Estimation of Difficulty

We can get an estimate of the total size of the search space for 10×10 domineering by estimating the total number of different board positions which are reachable. To estimate this we looked at the first 5 ply of the search space (see Figure 10). We note that on average the number of moves available decreases by about 3 with each ply of the search. This would mean that to search to the end of the game a depth of at least 30 would need to be reached. Also it can be noted that for the first five ply the effective branching factor ranges from 25 to 77. Therefore a conservative estimate of the effective branching factor at each ply would be a factor of 25. This data suggests that the size of the search space is approximately $25^{30} = 8.6 \cdot 10^{41}$.

Another method to determine the difficulty of the problem is to look at the number of nodes which it took to solve smaller boards and then just extend those numbers to the larger board. For example in the published results by Breuker *et al.* (2000) 7×7 domineering took about $4 \cdot 10^5$ nodes, 8×8 took $4 \cdot 10^8$ nodes, and 8×9 took $7 \cdot 10^{10}$ nodes. At this rate of growth we estimated that 9×9 could easily need more than $4 \cdot 10^{11}$ nodes and 10×10 domineering may take upwards of $5 \cdot 10^{14}$ nodes to solve. This is a number which is probably beyond our current computational resources.

The good news is that if we use this same method of estimation with the number of nodes which OBSEQUI needed to solve 8×8 domineering, $2 \cdot 10^6$, and 9×9 domineering, $2.5 \cdot 10^9$ nodes, we get a much smaller estimate of around $3 \cdot 10^{12}$ nodes to solve 10×10 domineering.

4.2 Splitting up the Work

Examining 3 trillion nodes is obviously much better than 500 trillion, but with only one processor we estimate that it would have taken upwards of 150 days, probably more since the transposition table would not have been sufficiently large to be effective. Therefore the work needed to be split up.

Our initial assumption was that 10×10 domineering was a first-player win. Therefore we ran our solver with the condition that every node at the eighth ply of the search was a first-player win. If our assumption of the values of these nodes was correct then this would be the first 8 ply, with transpositions removed, of the proof tree for 10×10 domineering. (See Figure 11). We then had the program write all the leaf nodes of this tree to a large file. Subsequently, the 650,531 nodes which were generated were split into a number of different work files, and each file was assigned to a different processor. Each of these processors worked on verifying that in fact all of these nodes were first-player wins.

Obviously, if any losses were found we would need to re-run our solver with the added knowledge of which cases were second-player wins, from which a new set of nodes would be generated. The process would iterate from there until we were able to find a set of leaf nodes which were all first-player wins. At this point we would have a correct proof for 10×10 domineering.

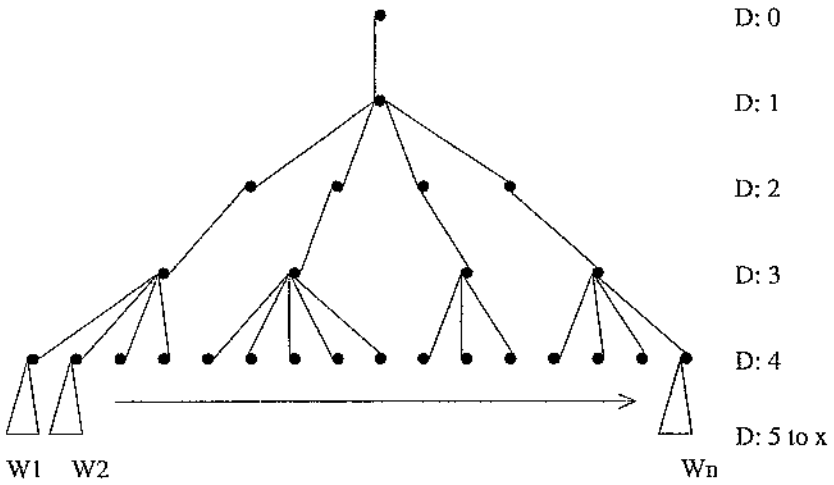

Figure 11: Minimal Alpha-Beta Tree of Depth 4 (assuming first-player win). Since we are assuming a first-player win, the first player only needs to make one move at each of his turns, while all of the second player's possible moves need to be examined. The small triangles at the bottom of the tree represent the independent searches that can be divided among many different processors to validate that the values of the nodes at the fourth ply of the tree are all wins for the first player.

4.3 10×10 is a First-Player Win

The computers which we used to solve this problem varied from 600 MHz Pentium 3 machines to 900 MHz Pentium 3 machines. Transposition-table sizes varied from 1 million to 8 million entries, depending on the amount of memory available on the given machine. The number of nodes which they could examine per second ranged from 140,000 to 210,000.

3,541,685,253,370 (3.5 trillion) nodes later the results were in: 10×10 domineering is indeed a first-player win. If Vertical is the first player then a winning move is position (1,2). OBSEQUI's original move ordering had been good enough that there was no need for a second iteration of the solving process: all 8-ply positions in the tree were proven to be wins.

4.4 Correctness

Computer programs may have subtle programming errors, therefore we would like to give a couple of reasons why we feel that the results can be trusted.

First, we have examined hundreds of random positions in the search trees. These positions have been examined to make sure that the right set of moves were generated, to make sure that the evaluation function returned the correct value for the given position, and to make sure the correct board positions were generated when we applied these moves to the current board position. Second, we ran OBSEQUI on all of the boards which have been previously solved by other researchers, and verified that OBSEQUI returns the same values for each of these boards. Third, in solving 10×10 domineering, we tried to make our method of breaking the problem up into smaller sub-problems as simple as possible, so as not to introduce new complexities which may have errors attached to them.

One final point we would like to address is the fact that OBSEQUI did not make any mistakes for the first player in any of their first four moves. This may seem surprising, but for example in 8×8 domineering, OBSEQUI did not make any mistakes until the sixth ply of the search, and then only made 2 mistakes out of 9862 moves. Therefore given the fact that the search space for 10×10 is considerably bigger, it is completely believable that OBSEQUI was able to extend its perfect accuracy to the eighth ply of the search.

For all of the reasons stated above we are very confident in our results. But as always independent confirmation of our 10×10 results would definitely be welcome and lend even greater credibility to the results.

5. CONCLUSIONS

The improvements made in the evaluation function and the use of the dominance relation have made an enormous impact on the size of domineering positions we are able to solve. (We also made some improvements to the move-ordering scheme and the transposition-table replacement scheme, see Bullock (2002)).

One measure of how far we have progressed is the length of time it takes to solve a specific board position. In "Solving 8 × 8 Domineering" by Breuker *et al.* (2000), submitted in 1998, it was mentioned that it took them 600 hours and almost 71 billion nodes to determine that 8 × 9 domineering was a win for the vertical player. OBSEQUI is able to tell us the same result in less than twenty minutes and needs to examine fewer than 260 million nodes (See Figure 12). This is a huge improvement in a time span of approximately 4 years, and represents a difference which is far larger than just the improvements in computer hardware.

A second measure of the progress which has been made is the number of new board positions which OBSEQUI can solve (See Figure 13).

Board Size	Result	Nodes	Board Size	Result	Nodes
2 × 2	I	1	4 × 7	V	802
2 × 3	I	2	4 × 8	H	2,570
2 × 4	H	7	4 × 9	V	13,570
2 × 5	V	6	5 × 5	2	259
2 × 6	I	8	5 × 6	H	324
2 × 7	I	2	5 × 7	H	2,210
2 × 8	H	26	5 × 8	H	2,467
2 × 9	V	65	5 × 9	H	11,669
3 × 3	I	1	6 × 6	I	908
3 × 4	H	5	6 × 7	V	24,227
3 × 5	H	14	6 × 8	H	204,813
3 × 6	H	16	6 × 9	V	1,374,535
3 × 7	H	43	7 × 7	I	31,440
3 × 8	H	33	7 × 8	H	949,209
3 × 9	H	100	7 × 9	H	6,052,516
4 × 4	I	23	8 × 8	I	2,023,301
4 × 5	V	42	8 × 9	V	259,064,428
4 × 6	I	583	9 × 9	I	1,657,032,906

Figure 12: Game-theoretic values for various sizes of domineering boards and the number of nodes OBSEQUI needed to calculate the values. (We used a transposition table with 2^{23} entries.)

Board Size	Result	Nodes
4 × 19	H	314,148,901
4 × 21	H	3,390,074,758
6 × 14	H	1,864,870,370
8 × 10	H	4,125,516,739
10 × 10	I	3,541,685,253,370

Figure 13: New game-theoretic values which have been determined by OBSEQUI for various interesting board positions, as well as the number of nodes needed to calculate the values. To solve the above positions we used a transposition table with 2^{23} entries except for 10 × 10 which was solved as described in Section 4.

5.1 Evaluation Function

The most significant improvement in our solver was our evaluation function. This enabled OBSEQUI to prune lines of search from its proof trees far faster than previous programs. This means that when a losing move is made our enhancements are able to determine this much sooner and therefore prune the subtree of that position.

In the 8×8 proof tree this enables our program to examine 40 times fewer nodes than it would without these enhancements. For example, with all of OBSEQUI's enhancements turned on it takes 2,023,301 nodes to prove $[8 \times 8] - 1$ ($[m \times n]$ denotes the game theoretic value of the $m \times n$ board). If we turn off all the enhancements to the evaluation function it takes 84,034,856 nodes. See Figure 14 and Figure 15 for a more detailed look at how individual enhancements affect the size of the proof tree for 8×8 domineering.

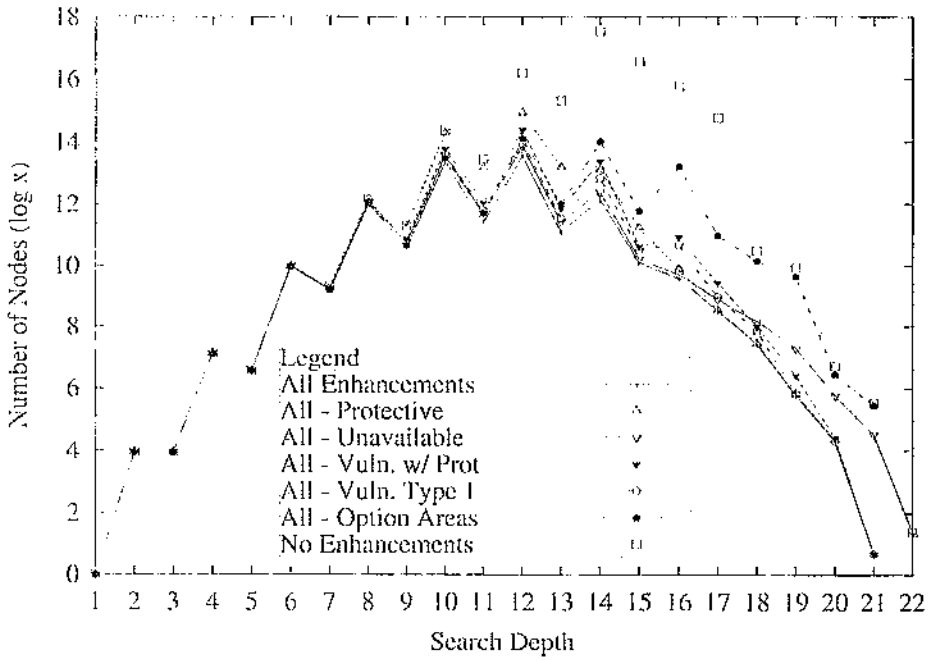

Figure 14: Comparison of different evaluation functions (number of nodes at each ply of the search). The x axis is the depth or ply of the search, the y axis is the log of the number of nodes which were examined at that depth of the search. "All - A" denotes that we used all of the enhancements except A.

Enhancements	Size of Proof Tree (nodes)
All Enhancements	2,023,301
All - protective areas	6,610,775
All - unavailable squares	2,566,004
All - vulnerable areas w/ protected squares	4,045,384
All - vulnerable type 1 areas	2,972,216
All - option areas	4,525,704
No Enhancements to Evaluation Function	84,034,856

Figure 15: Comparison of the size of the proof trees for 8×8 domineering given a certain evaluation function. "All - A" denotes that we used all of the enhancements except A.

5.2 Extending To All Rectangular Boards

Even with all the progress which has been made in solving the game of domineering we are still only able to solve a fairly small number of domineering boards. Thanks to the results of Lachmann *et al.* (2000) we are able to extend the results obtained through search techniques to much larger boards.

The new results which we have computed, in conjuction with the rules discovered by Lachmann *et al.* (2000), have enabled us to determine the values for a number of more board positions. Some of the more interesting values which we have obtained and which were not previously known, are:

- $[4 \times 19] = [4 \times 21] = H$. With these two results we now are able to determine who wins on all $4 \times x$ boards. $[4 \times x] = H$ for all values of x, where $x \geq 14$.
- $[6 \times 14] = H$. With this new result we can now determine $[6 \times x] = H$ for all even values of x, where $x \geq 20$.
- $[8 \times 10] = H$. This is the first $8 \times x$ board which has been determined to have a value of H.
- $[10 \times 10] = 1$. This is the largest square board ever solved.

See Figure 16 in the Appendix, for an updated table of who wins on rectangular boards.

6. ACKNOWLEDGMENTS

I would like to thank Martin Müller and Jonathan Schaeffer for all their feedback and suggestions. The financial support of Alberta's Informatics Circle of Research Excellence (iCORE) is also greatly appreciated.

7. REFERENCES

Berlekamp, E. R. (1988). Blockbusting and Domineering. *Journal of Combinatorial Theory, Series A*, Vol. 49, pp. 67–116. ISSN 0097–3165.

Berlekamp, E. R., Conway, J. H., and Guy, R. K. (1982). *Winning Ways for Your Mathematical Plays*. Academic Press, New York. ISBN 0–12–091150–7.

Breuker, D. M., Uiterwijk, J. W. H. M., and Herik, H. J. van den (2000). Solving 8×8 Domineering. *Theoretical Computer Science*, Vol. 230, pp. 195–206. ISSN 0304–3975.

Bullock, N. (2002). Domineering: Solving Large Combinatorial Search Spaces. M.Sc. thesis, University of Alberta. http://www.cs.ualberta.ca/~games/domineering.

Conway, J. H. (1986). *On Numbers and Games*. New Edition 2001. A.K. Peters, Ltd., Natick, MA, USA. ISBN 1–56881–127–6.

Gardner, M. (1974). (Mathematical Games). *Scientific American*, Vol. 230, No. 2, pp. 106–108. ISSN 0036–8733.

Herik, H. J. van den, Uiterwijk, J. W. H. M., and Rijswijck, J. van (2002). Games Solved: Now and in the future. *Artificial Intelligence*, Vol. 134, Nos. 1–2, pp. 277–311. ISSN 0304–3975.

Lachmann, M., Moore, C., and Rapaport, I. (2000). Who Wins Domineering on Rectangular Boards. *MSRI Workshop on Combinatorial Games* (ed. R. Nowakowski), pp. 307–315, Cambridge University Press, Cambridge, MA.

Uiterwijk, J. W. H. M. (2001). Personal communication.

8. APPENDIX

Below we provide an updated table of who wins on rectangular boards.

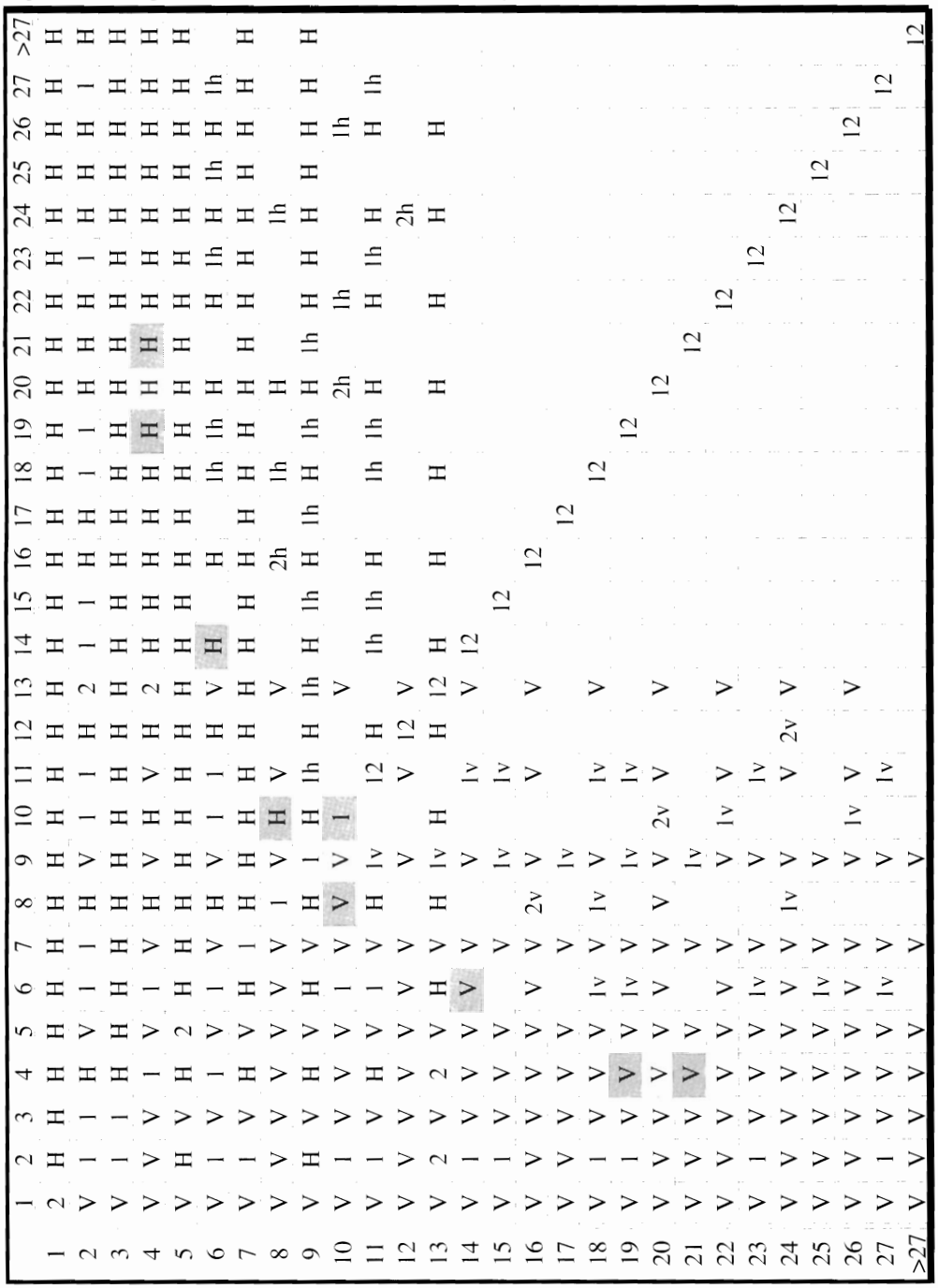

Figure 16: Updated chart of what we know about who wins Domineering on rectangular boards. New results which we have obtained from OBSEQUI are shaded. The y axis is the number of rows, x axis is the number of columns. A value such as 1h means the position is either a first-player or horizontal win (further work needs to be done to determine the exact value).