Notes on solving and playing peg solitaire on a computer

George I. Bell
gibell@comcast.net

Abstract

We consider the one-person game of peg solitaire played on a computer. Two popular board shapes are the 33-hole cross-shaped board, and the 15-hole triangular board—we use them as examples throughout. The basic game begins from a full board with one peg missing and finishes at a board position with one peg. First, we discuss ways to solve the basic game on a computer. Then we consider the problem of quickly distinguishing boards positions that can be reduced to one peg ("winning" board positions) from those that cannot ("losing" board positions). This enables a computer to alert the player if a jump under consideration leads to a dead end. On the 15-hole triangular board, it is possible to identify all winning board positions (from any single vacancy start) by storing a key set of 437 board positions. For the "central game" on the 33-hole cross-shaped board, we can identify all winning board positions by storing 839,536 board positions. Using these winning board positions, we calculate that the total number of solutions to the central game is 40,861,647,040,079,968.

1 Introduction

Peg solitaire is one of the oldest puzzles, with a 300 year history. The puzzle consists of a game board together with a number of pegs, or more commonly marbles. The board contains a grid of holes in which these pegs or marbles are placed. Figure 1 shows the most common shapes for a peg solitaire board, the 33-hole cross-shaped board, and the 15-hole triangular board.

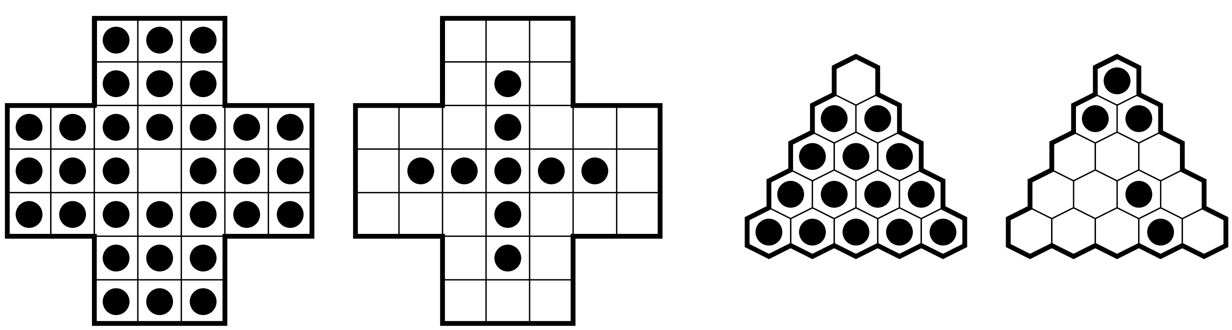


Figure 1: Popular peg solitaire boards: (a,b) the 33-hole cross-shaped board, (c,d) the 15-hole triangular board.

The game is played by jumping one peg over another into an empty hole, removing the peg that was jumped over. On the cross-shaped board of Figure 1a, these jumps must be made along columns or rows, whereas on the triangular board of Figure 1c, jumps are allowed along any of the six directions parallel to the sides of the board. The goal is to finish with one peg, a more advanced variation is to finish with one peg at a specified hole. The basic game begins from a full board with one peg removed, as in Figure 1a or c. Starting from Figure 1a and ending with one peg in the center of the board is known as the "central game" [1]. In Figure 1b or d we show starting configurations to be referred to later.

Only relatively recently in the history of the game have computers been used as an interface to play the game, as well as solve the game. There are now dozens of versions of the game available for playing on your computer or even your cell phone.

A computer version of the game is in many ways less satisfying than a physical game. However, there are some definite advantages to playing the game on a computer. The board can be reset instantly, and you won't be chasing marbles that fall off the board! You can take back a move, all the way back to the beginning if desired. This tends to make the game easier as you can more easily backtrack from dead ends. The sequence of moves leading to a solution can be recorded and played back. The computer can also be programmed to tell the user if the jump they are considering leads to a dead end or not. Adding this ability to a computer version of the game is tricky, and most versions do not have this ability. It is the goal of this paper to describe efficient techniques to enable a computer to point out all good and bad jumps from the current board position.

2 Board types and symmetry

We label the holes in the 33-hole board using Cartesian coordinates (Figure 2a), but with *y increasing* downward. For the triangular boards, we use "skew-coordinates" as shown in Figure 2b. By adding 1 to each coordinate, and converting the first to a letter, we obtain the standard board labellings used by Beasley [1] and Bell [4] (for example the central hole $(3,3)$ in Figure 2a becomes "d4").

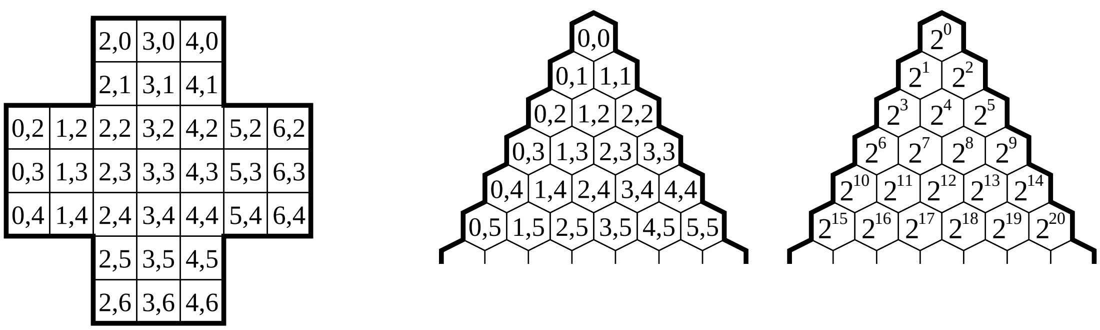


Figure 2: Hole coordinates for (a) the 33-hole cross-shaped board, (b) the triangular boards. (c) The weighting of each hole to convert a board position to binary.

The 33-hole board has square symmetry. There are eight symmetry transformations of the board, given by the identity, rotations of 90°, 180° and 270°, and a reflection of the board followed by these 4 rotations [6] (the dihedral group D_4). The triangular boards have 6-fold symmetry, with 3 possible rotations of 0°, 120° or 240°, plus a reflection followed by a rotation (the dihedral group D_3).

To store a particular board position on a computer, we convert it to an integer by taking one bit per hole. The most obvious way to do this is to take the board, row by row, top to bottom, as in Figure 2c. We will use N to denote the total number of holes on the board (the board size), so each board position is represented by an N bit integer. If b is a board position we'll denote this integer representation by $\text{code}(b)$. Most computer languages use a 32-bit integer, so we have one bit too many for storing the 33-hole board. Beasley [1, p. 249] gives a technique for storing a board position on the 33-hole board using 4 fewer bits. This technique is used in the ancillary program "pegs.cpp". For boards with more than 32 holes, we usually split $\text{code}(b)$ into several 4-byte integers.

The **complement** of a board position b is obtained by replacing every peg by a hole (i.e. removing it), and replacing every hole by a peg. The complement of b will be denoted as $\overline{b}$. We note that $\text{code}(\overline{b}) = \text{code}(f) - \text{code}(b)$, where f is the board position where every hole contains a peg, $\text{code}(f) = 2^N - 1$. The starting position for the "central game" in Figure 1a therefore has code $2^{33} - 2^{16} - 1$.

Two board positions are symmetry equivalent if one can be converted to the other by a symmetry transformation. This equivalence relation introduces a set of *equivalence classes* of board positions, which we call **symmetry classes**. The symmetry class of a board position does not change after it is rotated or reflected. One way to choose a representative from each symmetry class is to take the one with the smallest code. We use the notation $\text{mincode}(b)$ to denote this operation. For example, the board position b in Figure 1d has code $1+2+2^2+2^8+2^{13} = 8455$, and the other 5 codes obtained by symmetry transformation are: 2183, 3156, 3904, 25106 and 25280, so $\text{mincode}(b) = 2183$. We also have $\text{mincode}(\overline{b}) = (2^{15} - 1) - \text{maxcode}(b) = 7487$.

3 Single vacancy to single survivor problems

A peg solitaire problem which begins with one peg missing, with the goal to finish with one peg, will be called a **single vacancy to single survivor** problem, abbreviated SVSS. When the starting hole (x_0, y_0) and finishing hole (x_1, y_1) are the same, the SVSS problem is called a **complement problem**, because the starting and ending board positions are complements of each other.

A simple parity argument gives a necessary condition for solvability of a SVSS problem [1, Chapter 4]. On a square lattice (like the standard 33-hole board), the requirement is that x_0 and x_1 must differ by a multiple of 3, or that $x_0 \equiv x_1 \pmod 3$, and $y_0 \equiv y_1 \pmod 3$. Starting and ending board positions satisfying the above conditions are said to be in the same **position class**. The main result is that *the position class does not change as the game* *is played*. On a triangular lattice, the requirement is weaker: $x_0 + y_0 \equiv x_1 + y_1 \pmod 3$[1]. We will not go into the theory of position classes, the reader should see Beasley [1, Chapter 4] or Bell [4] for triangular peg solitaire.

It is interesting to see what happens to the position class after the board is rotated or reflected. For the central game on the 33-hole board (Figure 1a), the position class is not changed by rotations or reflections of the board. If we begin with $(3, 3)$ vacant we can finish at $(3, 3)$, or the rotationally equivalent holes $(3, 0)$, $(0, 3)$, $(6, 3)$ and $(3, 6)$. Any board position which begins from any of these five holes is in the position class of one peg in the center. Moreover, if we reflect and/or rotate the board at any time during the game, it remains in the same position class. Thus, this set of holes forms a set of SVSS problems of the same **type** whose general solutions are all interconnected, shown in Figure 3a.

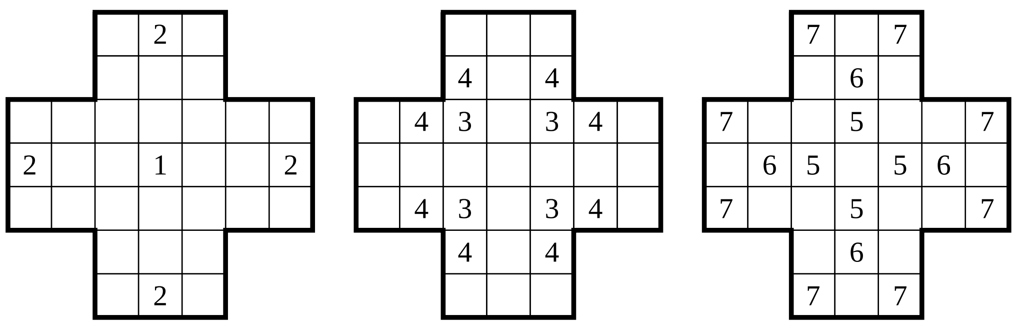


Figure 3: The three types of SVSS problem on the standard 33-hole board.

More commonly, the position class does change after the board is rotated or reflected. Problems of this type are shown in Figure 3b. In this case when we rotate the board, the position class changes, but only among the 4 with single peg representatives in the holes shown in Figure 3b. Another way to look at Figure 3b is that we can begin with (x_0, y_0) at any "3" or "4", and finish at any "3" or "4", if we allow peg solitaire jumps plus rotations and reflections of the board. The third type of problem on this board is given in Figure 3c. The three problem types are in a sense completely separate—it is never possible to move from a SVSS problem of one type to another, even if you are allowed to rotate or reflect the board.

Wiegleb's board (Figure 4) is an extension of the standard 33-hole board and has 45 holes [1, p. 199–201]. Figure 4 shows that this board also has three types of SVSS problems, but there are more of them (36 in all, see [2]). Figure 5 shows that the 6×6 square board also has three problem types.

In triangular peg solitaire the situation is somewhat different. On the triangular board of side n, if $n \equiv 1 \pmod 3$, the board is not null-class and no complement problem can be solved (see Bell [4]). The only SVSS problem on the 10-hole triangular board that is solvable is of the form: vacate $(0, 1)$ finish at $(1, 1)$. This gives only one type of problem that

[1] Both these conditions assume that the full and empty boards are in the same position class, a board satisfying this is called a **null-class** board (see Beasley [1]). All the boards we will consider are null-class, except for the triangular boards of side $4, 7, 10, \ldots$.

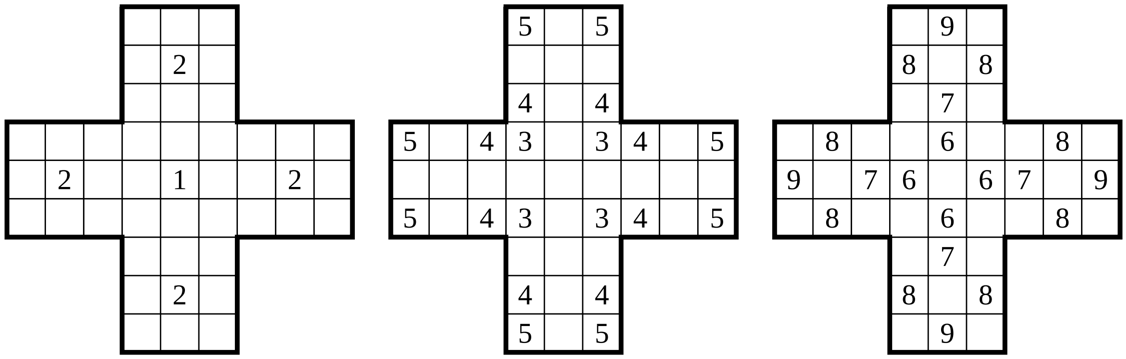


Figure 4: The three types of SVSS problem on Wiegleb's 45-hole board.

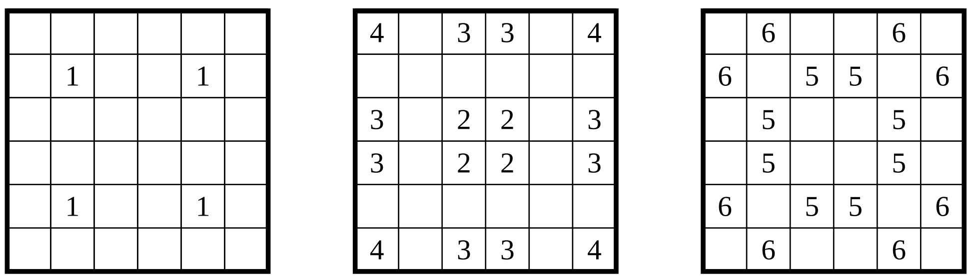


Figure 5: The three types of SVSS problem on the square 6x6 board.

is solvable (see Figure 6a), and any problem starting from an unmarked hole in Figure 6a cannot be solved to one peg (even if you are allowed to rotate and flip the board).

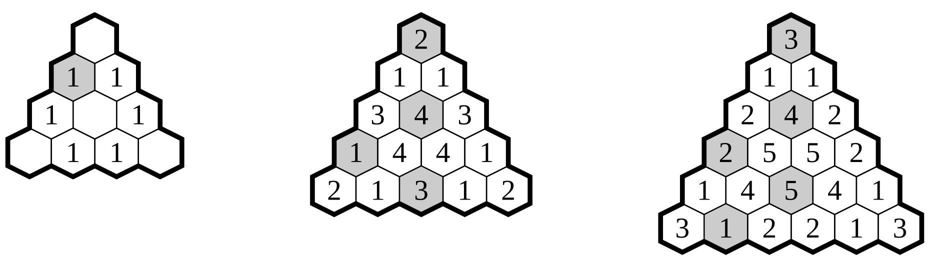


Figure 6: Types of SVSS problems on the triangular boards of side 4,5 and 6. The shaded holes show the "standard" starting vacancy for each problem number.

The 15-hole triangular board is null-class (Figure 6b). Here there are 4 different starting locations, the "standard starting holes" for each of the 4 are shaded. The fact that there is only one type indicates that all problems on this board are all interrelated.

The 21-hole triangular board is also null-class (Figure 6c), and contains 5 different starting locations. On this board it is possible to start with any peg missing, and finish at any hole using peg solitaire jumps, *plus* rotating and flipping the board.

Here are several peg solitaire problems we are interested in solving:

1. The **complement problem**: From a starting vacancy (x_0, y_0), execute an arbitrary number of jumps, then determine if the board position can be reduced to one peg at (x_0, y_0).
2. The **general SVSS problem**: From a starting vacancy (x_0, y_0), execute an arbitrary number of jumps, then determine if the board position can be reduced to one peg (anywhere on the board).
3. The **general problem**: From a given board position, determine if it can be reduced to one peg (anywhere on the board).

We will consider primarily the first two problems in this paper. We also want to solve these problems quickly—ideally within a web browser. In determining if a jump leads to a dead end or not, a delay of one second is unacceptable.

It is important to realize that problems #2 and #3 are different. For example, on the 33-hole board, a popular problem to solve is "cross" (Figure 1b). This board position is solvable to one peg, but can never appear in the solution to any SVSS problem. How do we know this? Because the complement of this board position cannot be reduced to one peg. See Bell [3] to clarify why SVSS board positions must have this property.

A fair question is, why not go for the most general and difficult problem #3? The reason is that the complement problem #1 and general SVSS problem #2 are significantly easier, because we can take advantage of special properties of their solutions.

3.1 Computer solving techniques

The simplest technique for solving a peg solitaire problem on a computer is to store the sequence of jumps, together with the current board position. One then performs a depth-first search by jump (extending the jump sequence and backtracking when no further move is possible). The 15-hole triangular board can be easily solved using this technique, but it is much slower on the 33-hole board. The reason is that there are a large number of jump sequences that result in the same board position, so there is a tremendous amount of duplicated work. This difficulty can be removed by storing board positions seen previously in a hash table or binary tree.

A better technique is to stop recording the jump sequence entirely, and look at the whole problem as a sequence of board positions. Given a set of board positions A, we denote by $D(A)$ the set of board positions that can be obtained by performing every possible jump to every element of A. We call $D(A)$ the **descendants** of A. As a programming task calculating $D(A)$ is straightforward. For example the set A can be stored on the disk as a sequence of integer codes, we read each code and convert it to a board position. From this board position we execute every possible jump, resulting in a large number of board positions which are stored in a binary tree (or hash table) to remove duplicates. This binary tree can be dumped to a file as a sequence of codes, the set $D(A)$.

The problem that eventually occurs is that the binary tree becomes too large to fit into memory[2]. At this point the problem is easily split into p smaller pieces that are calculated separately. It is best if p is a prime number, chosen so as to reduce the problem size so that it fits in memory. We now go through the board positions in A as before, but instead of storing each board position b in a binary tree, we convert it to a code and write that code into a temporary file numbered $\text{code}(b)\%p = \text{code}(b) \pmod p$. Here the "%" operator represents the remainder upon division by p (as in C). After all board positions in A have been considered, we now go back through each of the p temporary files, filling a binary tree for each p to remove duplicates and finally writing the unique board positions to the disk (in p separate files).

In most cases we do not want to store two board positions that are in the same symmetry class. For example, for the central game (Figure 1a, there are four first jumps, but these result in identical board positions which are rotations of one another. A nice way to select a single representative from the symmetry class is to use the one with the smallest code. In the above algorithms, we use $\text{mincode}(b)$ in place of $\text{code}(b)$.

For a set of board positions A, we denote by $|A|$ the number of elements in the set. $\bar{A}$ is the set of complemented board positions. In other words $b \in \bar{A}$ if and only if $\bar{b} \in A$. We note that $\bar{A}$ is not a set complement in the traditional sense (i.e. $b \in \bar{A}$ if and only if $b \notin A$), but rather *a set of complemented board positions*.

Let b_0 be the initial board position with one peg missing. Let b_1 be the final board position with one peg. Let $F_{N-1} = \{b_0\}$ and $B_1 = \{b_1\}$ (recall that N is the size of the board). We then define the set of board positions that can be reached from b_0 by:

$$F_n = D(F_{n+1}), n = N-2, N-3, \ldots, 1 \tag{1}$$

Note that every element of F_n has exactly n pegs. This produces a "playing forward" sequence of sets $F_{N-2}, F_{N-3}, \ldots, F_1$ where the number of elements in each set increases exponentially, at least initially. We are calculating the nodes in the "game tree", but have lost all information about the links connecting them (however, this link information is easily recovered). The problem has a solution if and only if $b_1 \in F_1$.

A sequence of sets can also be obtained from the finishing board position b_1 by "playing backwards", which in our notation we write as:

$$B_n = \overline{D\left(\overline{B_{n-1}}\right)}, n = 2, 3, \ldots, N-1 \tag{2}$$

Again, every element of B_n has exactly n pegs, and the problem has a solution if and only if $b_0 \in B_{N-1}$. It is worth noting that the sets B_n contain *every* board position which can be reduced to b_1. Thus, the sets B_n can be used to solve any problem #3 (p. 6) which finishes at b_1, or symmetric equivalents. If we calculate B_n over all possible one peg finishes, we can solve any problem #3.

The set of "winning board positions" with n pegs is defined as

$$W_n = F_n \cap B_n. \tag{3}$$

[2]On Wiegleb's 45-hole board, the tree reaches a size of 2 billion elements.

If we have *any solution*, and play this solution until reaching board b with n pegs, then it must be that $b \in W_n$. The sets W_n are usually much smaller than the F_n and B_n, these are the nuggets of gold that we seek, because they enable us to quickly recognize any winning board position.

As a practical matter, to find W_n it is not necessary to calculate every F_n and B_n for each n between 1 and $N-1$ and perform their intersection (intersecting two sets with potentially billions of elements is not a trivial computation). Suppose we can calculate the forward sets to F_k, and the backward sets to B_k for some k between $N-1$ and 1. If the problem has a solution, then $W_k = F_k \cap B_k$ is not empty. We then compute $W_{k-1}, W_{k-2}, \ldots, W_1 = \{b_1\}$ recursively using

$$W_n = D(W_{n+1}) \cap B_n \text{ for } n = k-1, k-2, \ldots, 1, \tag{4}$$

and $W_{k+1}, W_{k+2}, \ldots, W_{N-1} = \{b_0\}$ using

$$W_n = F_n \cap \overline{D(\overline{W_{n-1}})} \text{ for } n = k+1, k+2, \ldots, N-1. \tag{5}$$

To calculate using equation (5), we take each element of W_{n-1}, complement it, calculate all descendants and complement each. This yields the set $\overline{D(\overline{W_{n-1}})}$, and we now save each element which is in common with F_n, giving us W_n. The recursive calculations (4) and (5) are much easier than calculating all F_n and B_n because the sets W_n tend to be orders of magnitude smaller. The determination of W_n using (4) and (5) is considerably faster than the initial task of calculating the sets F_k and B_k.

Finally, we note that the sets F_n, B_n and W_n can be defined in two subtly different ways. First, they can simply be sets of board positions. If b_0 is the starting position for the central game (Figure 1a), then $F_{32} = \{b_0\}$, and F_{31} has 4 elements which are rotations of one another. We will sometimes refer to these sets as "raw F_n". In most cases, however, we will consider F_n, B_n and W_n as sets of symmetry classes. Now the set F_{31} only has a single element, which can be taken as any representative of this symmetry class, and generally we choose the one with the smallest code(). These sets are called "symmetry reduced" F_n. If we refer to an unqualified F_n or W_n it can be assumed to be symmetry reduced.

4 The complement problem

For a complement problem, we have $b_0 = \overline{b_1}$, which implies

$$B_n = \overline{F_{N-n}} \tag{6}$$

$$W_n = F_n \cap \overline{F_{N-n}} \tag{7}$$

$$\boxed{W_n = \overline{W_{N-n}}} \tag{8}$$

Equation (8) states that *the sets of winning board positions are complements of one another*. This is a remarkable result, and tells us that for a complement problem, we only need to store half the winning board positions. In order to calculate W_n the work is halved as well, for we need only calculate the forward sets F_n down to $k = \lfloor N/2 \rfloor$. After performing the intersection (7), the remaining W_n are then calculated using Equation (4) or equivalently (5), which can be written as

$$W_n = D(W_{n+1}) \cap \overline{F_{N-n}}. \tag{9}$$

Is storing winning board positions the most efficient technique? During the start of a game, it does not seem so, because all board positions that can be reached are winning. Perhaps it is better to store "losing board positions", or positions from which a one peg finish at the starting hole cannot be reached?

We could define the set of "losing board positions" with n pegs as those elements of F_n which are not in W_n. A more efficient technique is to store *only* those losing board positions which are one jump away from a winning board position. Thus, we define

$$L_n = D(W_{n-1}) - W_n \tag{10}$$

Table 1 shows the sizes of F_n, W_n and L_n for the 15 and 21-hole triangular boards. All winning board positions for any corner complement problem can be identified by storing just 95 board positions (15-hole board) or $26,401$ board positions (21-hole board). If we store losing board positions as defined by Equation (10), we need to store more than four times as many board positions.

For the corner complement problem on the 15-hole triangular board, the winning board positions W_n^2 are listed in the last section of this document. Note that the subscript refers to the number of pegs, while the superscript refers to the number assigned this starting vacancy in Figure 6. Storing the sets W_n^2 gives us a simple technique for determining if we are "on track" to solve the corner complement problem. Suppose the current board state is b.

1. If b contains more than $\lfloor N/2 \rfloor = 7$ pegs, complement the board position. The board position now has n pegs where $1 \leq n \leq \lfloor N/2 \rfloor$.
2. Calculate mincode(b).
3. If mincode(b) $\in W_n^2$, then the complement problem can be solved from the current board position, otherwise it cannot. Notice that mincode(b) $\notin W_n^2$ *does not* necessarily imply that the board position cannot be solved to one peg, just not to one peg *at the starting location.*

We also show the above algorithm in pseudocode:

```
W[1][2] = {1} ! The set W_1^2
W[2][2] = {10}
W[3][2] = {28, 84}
W[4][2] = {23, 35, 27, 35, 988, 508, 460, 134, 62}
...
! board is the current board position
! side is the triangular board side (4,5, or 6)
```

	15-hole triangular board				21-hole triangular board		
	Raw	Symmetry reduced			Symmetry reduced		
n (pegs)	$\|F_n\|$	$\|F_n\|$	$\|W_n\|$	$\|L_n\|$	$\|F_n\|$	$\|W_n\|$	$\|L_n\|$
20					1	1	0
19					1	1	0
18					4	4	0
17					23	23	0
16					117	117	0
15					522	503	19
14	1	1	1	0	1,881	1,690	185
13	2	1	1	0	5,286	4,328	907
12	8	4	2	2	11,754	8,229	3,288
11	35	19	9	4	20,860	11,506	8,478
10	122	62	18	20	28,697	11,506	14,701
9	293	149	29	43	29,784	8,229	16,856
8	530	268	35	86	23,263	4,328	13,063
7	679	344	35	94	14,039	1,690	7,267
6	623	317	29	89	6,683	503	3,005
5	414	215	18	49	2,545	117	935
4	212	112	9	29	774	23	211
3	75	39	2	7	168	4	34
2	18	10	1	1	28	1	4
1	4	3	1	1	5	1	1
Total	3,016	1,544	† 95	425	146,434	† 26,401	68,954

Table 1: Size of F_n, W_n and L_n for the corner complement problem on the 15 and 21-hole triangular boards. † only half of the W_n need to be stored, due to Equation (8).

```
! i is the number of this complement problem W[n][i]
problemIsSolvable(board, side, i) {
  n = CountPegs(board)
  totholes = side*(side+1)/2
  if (n > totholes/2) then {
    mincode = 2^totholes - 1 - GetMaxCode(board)
    n = tot - CountPegs(board)
  }
  else mincode = GetMinCode(board)

  for (j=0; j<Size(W[n][i]); j++) {
    if (mincode==W[n][i][j]) return true
  }

  return false
}
```

On board with less than about 25 holes, this test can easily be executed in a browser. For example, when the user mouses over a peg, we can test out the jumps from this peg and report whether the jump is "good" or "bad", namely leads to a winning or losing board position. In the web tool I have created [5], the bad jumps are humorously indicated by turning a peg into a bomb.

We have also calculated W_n^1 for the central game on the standard 33-hole board (Figure 3a). Table 2 shows the size of F_n^1 and W_n^1 for the central game on the 33-hole board. These sets are large enough that the array search in the algorithm `problemIsSolvable()` is too slow, and must be replaced by a faster search algorithm for good real-time performance[3]. The set of $839,536$ board positions, stored in 4-byte integers, requires 3.2 Megabytes of memory. Table 3 shows results for the central game on Wiegleb's board (Figure 4a). The set of $89,558,705$ board positions W_1 to W_{22}, stored in two 4-byte integers, requires 680 Megabytes of memory.

n (pegs)	$\|F_n\|$	$\|W_n\|$	n (pegs)	$\|F_n\|$	$\|W_n\|$
32	1	1	16	3,312,423	230,230
31	1	1	15	3,626,632	204,992
30	2	2	14	3,413,313	162,319
29	8	8	13	2,765,623	112,788
28	39	38	12	1,930,324	68,326
27	171	164	11	1,160,977	35,749
26	719	635	10	600,372	16,020
25	2,757	2,089	9	265,865	6,174
24	9,751	6,174	8	100,565	2,089
23	31,312	16,020	7	32,250	635
22	89,927	35,749	6	8,688	164
21	229,614	68,326	5	1,917	38
20	517,854	112,788	4	348	8
19	1,022,224	162,319	3	50	2
18	1,753,737	204,992	2	7	1
17	2,598,215	230,230	1	2	1
Total				23,475,688	† 839,536

Table 2: Size of F_n and W_n for the central game on the 33-hole cross-shaped board. † only half of the W_n need to be stored, due to Equation (8).

We can now solve all complement problems on the 15-hole triangular board by calculating all W_n^i. If we do this, we discover two problems with this technique. The first is a degeneracy of the finishing hole with respect to the board symmetry, while the second storage inefficiency is that the sets W_n^i may not be disjoint for different values of i.

[3] In the attachments to this paper, the sets W_n^1 are given sorted. The simplest search technique is then a binary search of a sorted array.

n (pegs)	$\lvert F_n \rvert$	$\lvert W_n \rvert$	n (pegs)	$\lvert F_n \rvert$	$\lvert W_n \rvert$
44	1	1	32	3,702,227	348,705
43	1	1	31	10,160,129	741,102
42	3	3	30	25,647,378	1,483,185
41	11	10	29	59,620,492	2,788,600
40	60	54	28	127,737,457	4,898,948
39	297	236	27	252,239,569	7,981,238
38	1,427	900	26	458,623,402	11,958,747
37	6,459	3,007	25	766,145,054	16,344,138
36	27,317	9,056	24	1,172,139,707	20,224,817
35	106,347	24,990	23	1,635,783,432	22,532,441
34	379,537	64,182	22	2,073,430,928	(same)
33	1,238,520	154,345	Total	6,586,989,754	89,558,705

Table 3: Size of F_n and W_n for the central game on Wiegleb's board (Figure 4a). Some elements of F_n that cannot appear in W_n have been removed by use of a resource count [1].

4.1 The symmetry degeneracy

This problem concerns the way we have reduced the set of board positions by using the symmetry of the board. There is no problem in this regard to the corner vacancy, or the central vacancy on the 33-hole standard board or Wiegleb's board. But suppose we look at the SVSS problem on the 15-hole triangular board starting from $(0,3)$. According to the position class theory, the possible finishing locations are given in Figure 7a[4].

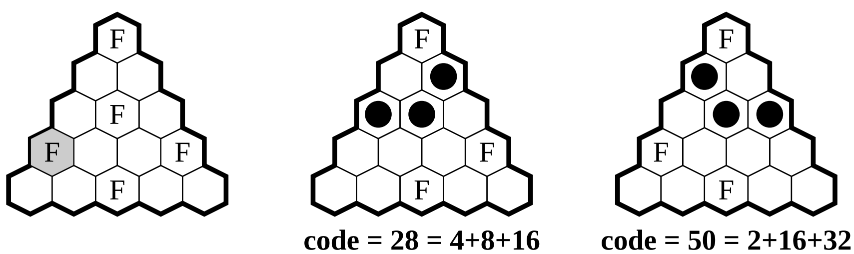


Figure 7: (a) Possible finishing holes starting with $(0,3)$ empty (shaded hole). (b) The $(0,3)$ finish can no longer be reached. (c) This board position is in the same symmetry class as (b), but the $(0,3)$ finish can be reached.

The problem occurs because it is possible to finish at the hole $(3,3)$, *and* that this hole is also mapped to the starting vacancy $(3,0)$ by a reflection of the board about the y-axis. The board positions in Figure 7b and 7c have the same mincode (28), because they are reflections of one another, and both can be reached starting from $(3,0)$. The problem is that we can't finish at $(3,0)$ from Figure 7b but we can from Figure 7c, yet according to our algorithm these board positions are "the same" (they lie in the same symmetry class).

[4]From [4], we know that it is not possible to finish at $(2,1)$, but this need not concern us here.

If we create the sets W_n^1 using the symmetry reduction technique of using mincode(b) all will work perfectly, except that our program will consider the finishing holes $(0,3)$ and $(3,3)$ to be the same. One resolution of this degeneracy is to loosen our definition of "complement problem" to include any finishing board position which is in the same symmetry class. In other words had we defined the problems we are trying to solve differently, the degeneracy disappears!

But let us assume we do not want to define the problem away, and stick with our definition of complement problem. To resolve the degeneracy we are forced to *not to do symmetry reduction of the sets*, leaving them in their *raw* state. We then lose the simple check of calculating mincode(b), and checking this against the sets W_n^1. Instead, we must figure out the symmetry transformation S which takes us from the starting board state to one peg missing at $(3,0)$. Given any board position b, we then check to see if code($S(b)$) is in the set W_n^1. Note that since we have not done symmetry reduction of the sets W_n^1, there can be two members of this set in the same symmetry class, so with the same mincode(). Unfortunately, this significantly complicates our algorithm for identifying winning board positions.

4.2 The storage inefficiency

We note from Figure 7 that the minimum code 28 must lie in W_3^1, W_3^2 and W_3^3 because this mincode() can appear during all three complement problems. This indicates that winning board positions for different complement problems will share members, and not just occasionally. In fact, W_n^1 and W_n^2 have almost all of their elements in common. This is not really a problem on the 15-hole triangular board, because these sets are small. It becomes more of a problem for the 21-hole triangular board, and the 33-hole cross-shaped board. We will discuss solutions to this problem in the next section.

5 The general SVSS problem

The key feature of solutions to complement problems that gives $W_n = \overline{W_{N-n}}$ is that the starting set $F_{N-1} = \{b_0\}$ and the finishing set $B_1 = \{\overline{b_0}\}$ are complements of one another. These two sets need not contain only a single board position. For example, let $\mathcal{F}_{N-1}$ be all board positions of the same problem type with one peg missing and $\mathcal{B}_1 = \overline{\mathcal{F}_{N-1}}$ all one-peg board positions of this type.

As before we have $\mathcal{W}_n = \mathcal{F}_n \cap \mathcal{F}_{N-n}$ and $\mathcal{W}_n = \overline{\mathcal{W}_{N-n}}$. The winning "superset" $\mathcal{W}_n$ contains all board positions that can be reached in the solution to a single vacancy to single survivor problem of this type. We see, in fact, that $\mathcal{W}_n$ is the union of all W_n^i over all complement problems i of a given type, plus a special set which we call W_n^0 containing all board positions which can occur in SVSS problems of this type but not in any complement problem. We then have

$$\mathcal{W}_n = \bigcup_{i=0,1,\dots,p} W_n^i \text{ where } W_n^0 \cap W_n^i = \varnothing, i = 1, 2, \dots, p \tag{11}$$

We note that since $\mathcal{W}_n = \overline{\mathcal{W}_{N-n}}$ and $W_n^i = \overline{W_{N-n}^i}$ for $i = 1, 2, \ldots, p$, it must be the case that $W_n^0 = \overline{W_{N-n}^0}$.

We have already seen how to calculate the complement problem sets W_n^1, W_n^2, …, W_n^p, but how can we calculate W_n^0? One way is to calculate the set $\mathcal{W}_n$ directly using all possible starting and finishing locations, and then subtract out each W_n^i for $i = 1, 2, \ldots, p$. This is certainly not difficult for a small board like the 15-hole triangle. Another technique is to calculate all board positions that can be appear in SVSS problems that are not complement problems, taking their union and then subtracting off the complement problems as before.

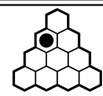
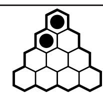
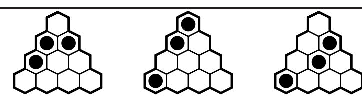
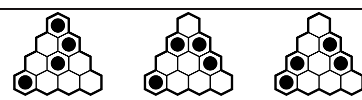
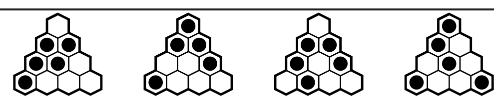
n (pegs)	$W_n^0 = \mathcal{W}_n$
1	 **code = 2**
2	 **code = 3**
3	 **code = 14** **code = 67** **code = 84**
4	 **code = 85** **code = 102** **code = 108**
5	 **code = 94** **code = 103** **code = 174** **code = 595**

Table 4: The sets $W_n^0 = \mathcal{W}_n$ for the 10-hole triangular board. All sets are symmetry reduced.

The 10-hole triangular board provides a simple example of the sets W_n^0. This board is not null-class, so no complement problem is solvable. What this means is that **all** sets W_n^i are empty for $i > 0$ (or that the number of solvable complement problems $p = 0$), so that the only sets around are $W_n^0 = \mathcal{W}_n$. These sets can be calculated quite easily (even by hand), and are shown in Table 4. Here we also see a board where the total number of holes $N = 10$ is even, so that $W_5^0 = \overline{W_5^0}$ is equal to its own complement. We can see that this is in fact the case, as the four board positions in W_5^0 are listed in pairs that are complements of one other. We must be careful to interpret the board positions in W_5^0 as symmetry classes, the complement of the board position with code 94 has code $2^{10} - 1 - 94 = 929$, a board position in the same symmetry class as the board with code 103. We conclude from Table 4 that only 10 essentially different board positions (or their complement) can appear during a solution to any SVSS problem on this board (we only need half of the set W_5^0).

6 The complement plus general SVSS problem

We now show how to solve either the complement problem (#1), or the general SVSS problem (#2), with very little additional storage over the general SVSS problem. Equation (11) shows that these two problems are closely related. The only difficulty involves storing board positions efficiently (without duplicates), and dealing with the degeneracies introduced in Section 4.1. In this section we use as an example the 15-hole triangular board.

We deal first with the storage problem. How can we store all board positions in W_n^i without duplication? Here we are considering all the problems of a certain type on a board, and $i = 1, 2, \ldots, p$ ranges over the total number of problems of this type. On the 15-hole triangular board, there is only one type (shown in Figure 6b) with $p = 4$ different problems. The obvious solution is to take all possible combinations of the 4 problems, $2^p = 16$ possibilities, and for each combination we store all boards common to this combination of problems. We can think of these as sets W_n with a superscript given not by the problem number i, but the *index* of the combination of complement problems that this board position can occur in (ranging from 0 to 16). Index 0 remains the same as problem 0: $W_n^0 = W_n^{index=0}$ is the set of n-peg board positions that can occur during a SVSS problem, but not in any complement problem.

	$\lvert W_n^{index}\rvert$ for $index = 0$ to 7							
Pegs (n)	0	1	2	3	4	6	7	Total
1	1	1	1	0	1	0	0	4
2	1	0	0	1	2	0	0	4
3	3	0	0	1	7	0	1	12
4	5	0	0	4	19	1	4	33
5	10	0	1	8	49	4	8	80
6	7	0	2	11	93	6	13	132
7	4	0	2	12	129	7	18	172
Total	31	1	6	37	300	18	44	437

Table 5: A count of winning board positions W_n^{index} on the 15-hole triangular board.

For example, since $index = 7 = 0111$ in binary, then $W_n^{index=7}$ contains all n-peg positions that are common to problems 1, 2 and 3. We note the since problem 4 is unsolvable as a complement problem [4], $W_n^4 = \varnothing$ and all sets W_n^{index} with $index$ between 8 and 15 are also empty.

A trickier question is how to resolve the degeneracy at the $(0, 3)$ starting location. The sets W_n^1 cannot be symmetry reduced, yet W_n^2 and W_n^3 are symmetry reduced, and $W_n^4 = \varnothing$. We see from Figure 7 that W_n^1 contains code 50, while W_n^2 contains mincode 28, a board position in the same symmetry class. The solution is to use an algorithm which keeps all codes in W_n^1 but removes all symmetry equivalents in the intersecting sets. This is the reason that the degenerate starting locations are indexed first.

Table 5 summarizes the number of board positions by *index* and number of pegs n. The total number of board positions over all sets is 437, which is only 10 more than are needed to solve the general SVSS problem by itself. We can infer that there are 10 board positions among the 437 that are duplicated—these must be 10 pairs of board positions in W_n^1 that are in the same symmetry class.

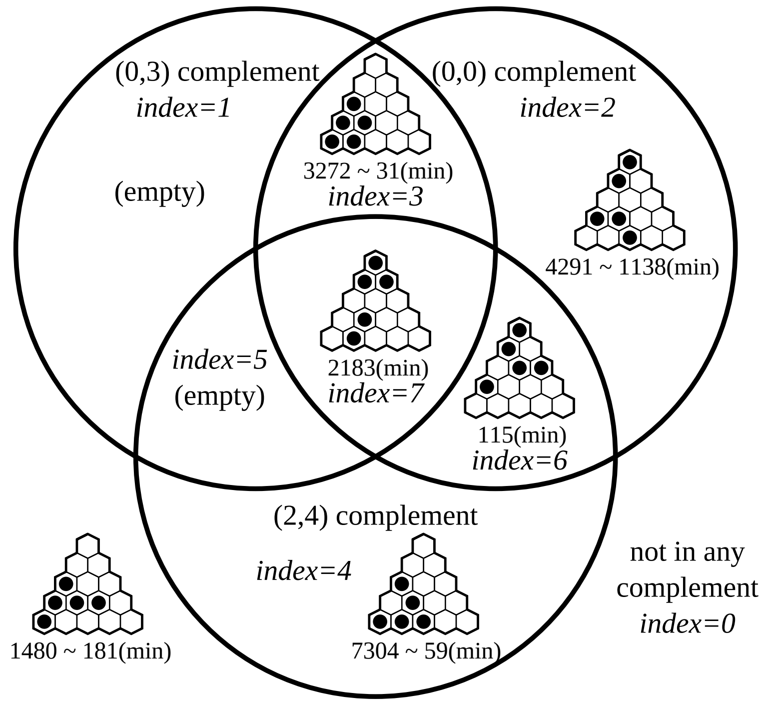


Figure 8: A Venn diagram showing sample boards with 5 pegs by index.

Figure 8 shows representative 5-peg board positions in W_5^{index} for values of the *index* 0 to 7. Let us interpret two of the board positions in this diagram. The board position with code 1480 is in *index* 0, meaning that this board position cannot appear in any complement problem. This board position has a mincode of 181. We can finish with one peg from this board position at $(0, 0)$ or $(0, 3)$, but we cannot start from either of these holes and reach this board position. But it must be possible to reach this board position from some start, and it turns out this start is $(2, 4)$.

The board position with *index* 6 is 115, which is the mincode. We can play from this board position to finish at $(0, 0)$, $(0, 3)$ or $(2, 4)$, and we can reach this board position from $(0, 0)$ or $(2, 4)$. Therefore, this board position can be reached during the solution to the $(0, 0)$ or $(2, 4)$ complements, so is in *index* 6.

We now present pseudocode for identification of winning board positions for either the complement problem (#1) or the general SVSS problem (#2):

```
W[5][1] = {16,64,1,8} ! W^1, index=0,1, ... 15, for the 15-hole triangular board
End[5][1] = {1,2,3,3,4,4,4,4,4,4,4,4,4,4,4,4} ! Ends of each index 0-15
... see Triangle15Winning.txt ...
```

```
! board is the current board position
! side is the triangular board side (4,5, or 6)
! i is the number of this SVSS problem
! ksym is the symmetry code of the starting board position
! comp is true for complement problems, otherwise any finish is assumed
boardIsSolvable(board, side, i, ksym, comp) {
  int code[6]

  totholes = side*(side+1)/2

  if (side==4) { ! 10-hole triangular board
    topindex = 1
    degen = 0
  }
  if (side==5) { ! 15-hole triangular board
    topindex = 2^4 ! power set of 4 problems
    degen = 1 ! number of degenerate problems
  }
  if (side==6) { ! 21-hole triangular board
    topindex = 2^5 ! power set of 5 problems
    degen = 2 ! number of degenerate problems
  }

  n = CountPegs(board)
  if (n > tot/2) then {
    code[0] = 2^tot - 1 - Code(board)
    n = tot - CountPegs(board)
  }
  else code[0] = Code(board)

  ! Get the 6 symmetry codes, code[0] to code[5]
  code[1] = rotatecode(code[0])
  code[2] = rotatecode(code[1])
  code[3] = reflectcode(code[2])
  code[4] = rotatecode(code[3])
  code[5] = rotatecode(code[4])

  if (comp) { ! complement problem
    kStart = 0
    kEnd = 6
    if (i<=degen) {
      kStart = kSym
      kEnd = kSym + 1
    }
    for (index=1; index<topindex; index++) {
      if ((1<<i) & index) { ! true if the i'th bit of index is set
        for (j=End[side][n][index-1]; j<End[side][n][index]; j++) {
          for (k=kStart; k<kEnd; k++) if (code[k]==W[side][n][j]) return true
        }
      }
    }
  }
  else { ! comp=false, finish anywhere
```

```
    for (j=0; j<End[side][n][topindex]; j++) {
      for (k=0; k<6; k++) if (code[k]==W[side][n][j]) return true
    }
  }

  return false
}
```

7 On-line games and software

I have created a Javascript game [5] for playing peg solitaire on the 10, 15 and 21-hole triangular boards. This game can begin from any starting vacancy, and the program will point out all jumps which lead to winning or losing board positions. The game can be specified as either a complement problem (the user must finish with one peg at the location of the original hole), or the general problem with a one peg finish anywhere on the board. The algorithm used to identify winning and losing board positions is the algorithm `boardIsSolvable()` in this paper converted to Javascript.

The ancillary files for this paper include the following files:

- `Triangle10Winning.txt` – a text file of the board positions $\mathcal{W}_n$, given in Table 4. Note: Each file `Triangle??Winning.txt` contains two versions of the solution sets $\mathcal{W}_n$. First, a version for identifying winning board positions in the "finish anywhere" problem (#2) using the algorithm `problemIsSolvable()`. Second, an *indexed* version as described in Section 6, which is slightly larger due to the symmetry degeneracy. These sets can be used to identify winning board positions for any complement problem (#1) as well as the "finish anywhere" problem (#2) using the more complex algorithm `boardIsSolvable()`. Each set is sorted by index (if present), then by code. On this 10 hole board the two versions are identical so the same sets appear twice.

- `Triangle15Winning.txt` – a text file of the board positions $\mathcal{W}_n$ for the 15-hole triangular board. This is given as a single set subdivided by index. The array `End[index]` (summarized by Table 5) shows where the end of the codes for each index occurs. The function `boardIsSolvable()` shows how this information is used.

  The sets for the corner complement problem (problem $i = 2$):
  $W_1^2 = \{1\}$, $W_2^2 = \{10\}$, $W_3^2 = \{28, 112\}$, $W_4^2 = \{23, 58, 85, 120, 1108, 1616, 2076, 2210, 2272\}$, $W_5^2 = \{31, 93, 115, 601, 1054, 1138, 1140, 1562, 1648, 2183, 2218, 2245, 2280, 2348, 2472, 2616, 2728, 2819\}$, $W_6^2 = \{125, 633, 1086, 1111, 1594, 1621, 2191, 2253, 2275, 2289, 2343, 2467, 2589, 2723, 2785, 2841, 2889, 3126, 3250, 3298, 3428, 3634, 3845, 4220, 4270, 4282, 4691, 4728, 4817\}$, $W_7^2 = \{1567, 1651, 2235, 2365, 2413, 2537, 2731, 2793, 3159, 3196, 3320, 3374, 3388, 3607, 3642, 3667, 3669, 3704, 3859, 3921, 4215, 4339, 4341, 4469, 4701, 4849, 5302, 5350, 5746, 5810, 6881, 6985, 10053, 10065, 12065\}$

- `Triangle21Winning.txt` – a text file of the board positions $\mathcal{W}_n$ for the 21-hole triangular board.

- `Triangular1.2` – a collection of html and javascript programs which can solve SVSS problems on triangular boards of arbitrary size.

- `NeverLose1.4` – a collection of html and javascript programs which can identify all winning positions on the 10, 15, and 21-hole triangular boards (source files for [5]).

- `pegs` – a collection of C++ routines for calculating winning sets W_n for the 33-hole cross-shaped board using Equation (9). Table 2 gives the sizes these sets for the central game. If printed out in a text file, all 839,536 winning board positions for the central game take up about 9MB. These can be printed out by the above program, the beginning and end of these sets is given below.

  $W_1 = \{65536 = 2^{16}\}$, $W_2 = \{528 = 2^4 + 2^9\}$,
  $W_3 = \{400 = 2^4 + 2^7 + 2^8, 212992 = 2^{14} + 2^{16} + 2^{17}\}$,
  $W_4 = \{153, 1680, 16688, 17928, 66432, 82976, 147984, 352256\}$,
  $W_5 = \{158, 692, 793, \ldots, 4554760, 6684688, 8601616\}$,
  ...
  $W_{14} = \{53247, 56831, 57279, \ldots, 2651879594, 2655805539, 3098292302\}$,
  $W_{15} = \{127999, 128895, 129791, \ldots, 3793449102, 3793531059, 3793629859\}$,
  $W_{16} = \{126975, 130559, 229359, \ldots, 3864553651, 3928764638, 3929805043\}$.

  The pegs directory also contains the program "count.cpp", which counts the number of winning games after "pegs.cpp" has been run.

8 Results from winning board calculations

8.1 How badly can you play?

After calculating the winning sets W_n we can determine the first possible dead-end. This is the shortest jump sequence after which the goal can no longer be reached. A well-known sequence of 4 jumps is a dead end for the central game (proved in [1, p. 115]). We have calculated the first dead end position for all seven complement problems, the results are shown in Table 6. Surprisingly, there are three complement problems that can be lost in only three jumps. All complement problems also have a unique move sequence leading to a dead end position (up to symmetry and move order), with the exception of the c3-complement. Moves are presented in the alphanumeric format used by John Beasley [1].

Table 6 indicates that a dead end for the d1-complement can be reached after only 3 jumps. If we start with the center vacant, we can reach the same board position B by playing d2-d4, and then the same second and third jumps. Doesn't this mean that a dead end for the central game can also be reached in 3 jumps? In fact it does not, for the d1-complement requires that we start *and* finish at d1. From board position B, we can no longer finish with one peg at d1, but we can finish with one peg at a4, g4, or d4. Thus, these three jumps are not a dead end for the central game.

Complement problem	Problem symmetry	$\sum \lvert W_n \rvert$	First Dead End
$(3,3)$ or d4	full	839,536	d2-d4, d5-d3, b4-d4, d3-d5 (4 jumps)
$(3,2)$ or d3	y-axis	6,420,923	d1-d3, d4-d2, d6-d4 (3 jumps)
$(3,1)$ or d2	y-axis	760,164	d4-d2, b4-d4, e4-c4 (3 jumps)
$(3,0)$ or d1	y-axis	99,982	d3-d1, b3-d3, e3-c3 (3 jumps)
$(2,2)$ or c3	diagonal	20,836,420	c1-c3, e2-c2, c3-c1, d4-d2, b4-d4, f3-d3 (6 jumps)
$(2,1)$ or c2	none	12,372,794	c4-c2, a4-c4, d4-b4, f4-d4 (4 jumps)
$(2,0)$ or c1	none	13,918,925	c3-c1, e2-c2, d4-d2, c2-e2, b4-d4 (5 jumps)

Table 6: Shortest dead end positions for all complement problems on the 33-hole board. All sets W_n are symmetry reduced.

8.2 How many wins are there?

We can also use W_n to count all solutions to the central game. Suppose we take the set of winning board positions, and add directed edges between board positions related by peg solitaire jumps. This results in a directed graph of the peg solitaire win, Figure 9 shows this graph for the 10-hole triangular board.

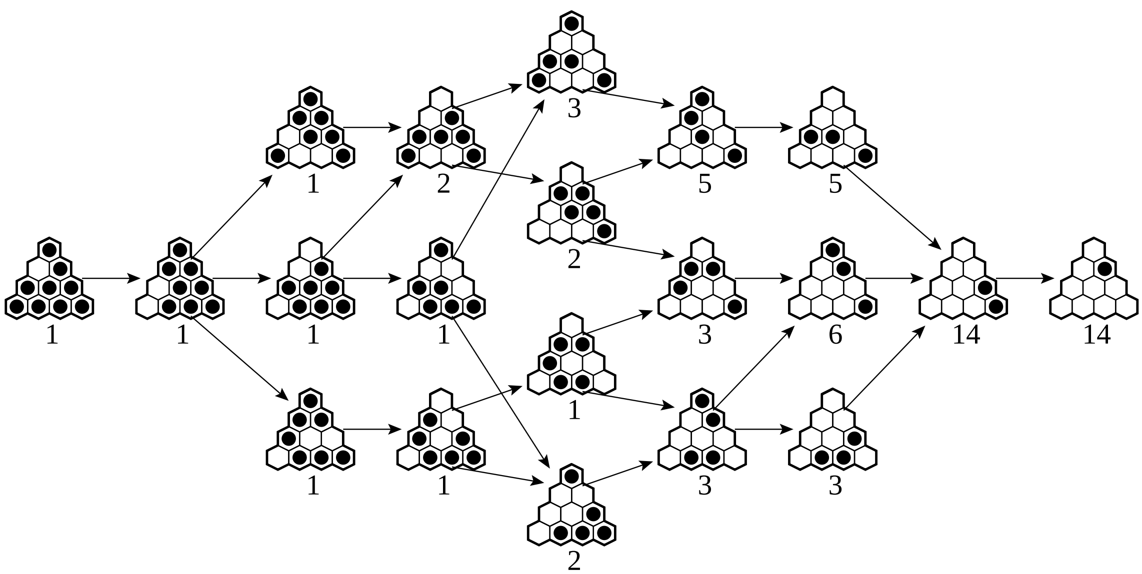


Figure 9: A winning game graph for counting all wins on the 10-hole triangular board.

The total number of wins is simply the total number of ways to traverse this graph. There is a simple algorithm for calculating this. We will label each vertex (board position) with a number, which will be the number of paths from the start to that board position. We first label the starting board position with a 1. We then consider all descendants of labeled board positions, and label each as the sum of all incoming edges. We continue this process until we reach the final board position. As can be seen in Figure 9, the total number of ways to traverse the graph, or the number of solutions, is 14.

We have run this counting algorithm on the central game on the 33-hole board, using the ancillary program "count.cpp". In the calculation we have eliminated symmetry considerations to be sure that we calculate every solution. This graph therefore has more than the $839,536$ nodes of the symmetry reduced winning sets. The program calculates that the total number of winning games is $40,861,647,040,079,968 \approx 4.086 \times 10^{16}$, in perfect agreement with the figure calculated by Bill Butler [7]. We have also counted solutions for all seven complement problems, see the comments in ancillary program "count.cpp".

9 Summary

We have introduced some techniques in peg solitaire that allow winning board positions to be identified. We accomplish this by storing a certain core set of board positions that can appear during single vacancy to single survivor problems. These board positions allow us to create a program that will take any SVSS program and identify all jumps that can lead to a one peg win, or conversely all dead ends where a solution can no longer be reached.

We note that these techniques, by design, only work *only* for beard positions which can appear during SVSS problems. Suppose we consider an *arbitrary* board position b. A reader may conclude from Equation (8) that if b is solvable to one peg, then $\bar{b}$ must also be solvable to one peg. **This is false!** Consider, for example, the board position "cross" Figure 1b. The statement of Equation (8) applies only to board positions which can arise during SVSS problems, not any arbitrary board position. This boils down to the difference between solving Problems 1 and 2 compared with Problem 3 as discussed on page 6 (Section 3).

We have also shown how the winning sets enable some interesting peg solitaire calculations. We calculated the first possible dead end position for all complement problems on the 33-hole board, and we counted all solutions to the central game.

References

[1] J. Beasley, *The Ins and Outs of Peg Solitaire*, Oxford Univ. Press, 1992.

[2] G. Bell and J. Beasley, New problems on old solitaire boards, Board Game Studies, **8** (2006), http://arxiv.org/abs/math/0611091.

[3] G. Bell, Triangular peg solitaire unlimited, *Games and Puzzles J.* #36 (2004), http://www.gpj.connectfree.co.uk/gpjr.htm and http://arxiv.org/abs/0711.0486.

[4] G. Bell, Solving triangular peg solitaire, *J. Integer Sequences*, Article 08.4.8, **11** (2008), http://arxiv.org/abs/math/0703865.

[5] G. Bell, "Never Lose" peg solitaire game, http://home.comcast.net/ gibell/pegsolitaire/Tools/Neverlose/Triang.htm

[6] G. Bell, A fresh look at peg solitaire, *Math. Mag.* **80** (2007), 16–28.

[7] B. Butler, http://www.durangobill.com/Peg33.html.