Some observations on the analysis of peg solitaire by computer

John Beasley, 23 / 27 February 2013, minor corrections 15 March

George Bell responded to my posting of *On 33-hole solitaire positions with rotational symmetry* elsewhere on this site by drawing my attention to a paper *Solving peg solitaire with bidirectional BFIDA** by Joseph K. Barker and Richard E. Korf, which was presented to the 26th AAAI Conference on Aritifical Intelligence last year.[†] In it, Barker and Korf comment on the usefulness of the bidirectional approach (searching back from the goal as well as forward from the starting position, and seeing where the two search trees meet) in reducing search time. This was contrary to my own experience when examining the game by computer in 1984-85, when I found that incorporating a backward search from the goal would be not merely unhelpful but even disadvantageous, and it has occurred to me to revisit my old calculations and to see why we came to such different conclusions.

Background

Peg solitaire is a peg-jumping game for one person which is normally now played on the 33-hole board shown in Figure 1, though in principle a board of any size and shape may be used and many other boards have been tried (not necessarily with the permitted lines of movement at right angles, nor even straight). We shall use an algebraic notation much as is used in chess, though chess players are asked to note that in solitaire we put row 1 at the top. The rule of play is to jump a peg over an adjacent peg into an empty hole immediately beyond, the peg jumped over being removed from the board, and the normal task is to start with the board full apart from one hole and to play so as to end with a single peg somewhere. Once this has been achieved, the game can be enriched in various ways, one of which is to regard a sequence of jumps by the same peg as a single move and to try and solve the game using the minimum number of separate moves, and it is with this form of the game that we shall concern ourselves here. The game has been discussed in many books and articles, in particular in chapter 23 of *Winning Ways for your Mathematical Plays* by E. L. Berlekamp, J. H. Conway, and R. K. Guy (1982, second edition 2004) and in my own *The Ins and Outs of Peg Solitaire* (1985, paperback edition 1992), but this presentation will be complete in itself and readers already familiar with *Winning Ways* or *The Ins and Outs* are asked to bear with a brief introduction to set the scene.

	a	*b*	*c*	*d*	*e*	*f*	*g*
1			○	○	○		
2			○	○	○		
3	○	○	○	○	○	○	○
4	○	○	○	○	○	○	○
5	○	○	○	○	○	○	○
6			○	○	○		
7			○	○	○		

Figure 1

	a	*b*	*c*	*d*	*e*	*f*	*g*
1			*tc*	*s*	*tc*		
2			*t*	*ok*	*t*		
3	*sc*	*s*	*ic*	*s*	*ic*	*s*	*sc*
4	*t*	*ok*	*t*	*ik*	*t*	*ok*	*t*
5	*sc*	*s*	*ic*	*s*	*ic*	*s*	*sc*
6			*t*	*ok*	*t*		
7			*tc*	*s*	*tc*		

Figure 2

In the form which we shall consider here, the game permits jumps in any horizontal or vertical direction, though not diagonally, and since any jump moves a man two holes the pegs are immediately divided into four classes as shown in Figure 2. One of these comprises the twelve *corner* pegs, which in turn are divided into the four *side corners* (*sc* in Figure 2), the four *top and bottom corners* (*tc* in Figure 2), and the four *inside corners* (*ic*). We may notice that while a corner peg remains a corner peg throughout its existence, its flavour may change during the play, and in particular a side or top and bottom corner peg may and indeed must move to an inside corner in order to be jumped over and removed. The second class comprises the eight *s pegs*, which again remain s pegs throughout their existence, and we note that any jump by a side corner peg to an inside corner in order to be captured must consume an s peg. The third class comprises the eight *t pegs*, which play a similar role in the clearance of the top and bottom corners, and the fourth class comprise the five remaining pegs. These are the pegs which can move to the centre and for this reason I called them "middle pegs" in *The Ins and Outs*, but I now consider this to have been a mistake and regard the old term *key pegs* as preferable. They in turn appear in two flavours, the four *outside key pegs* (*ok*) and the single *inside key peg* (*ik*), and while they always remain key pegs their flavour may change during the play.

[†] AAAI : Association for the Advancement of Artificial Intelligence.

Brute force and constraints

The simplest way of analysing a game such as peg solitaire, if the capacity of the available machine permits, is by an exhaustive enumeration of all possible positions. Since the board contains 33 holes and each can be either full or empty, this appears to require 2^{33} bits of memory, but the positions on a square lattice solitaire board can be divided into sixteen classes such that if a position is in a certain class then so are all its progeny.[‡] So, given a starting position, 2^{29} bits of memory suffice to define all the positions that might be derivable from it, and this is sufficient to allow an exhaustive enumeration to be made. However, all these 2^{29} bits must be held in immediate-access memory, otherwise the calculation is likely to be unacceptably slow.

An analysis which takes account of all 2^{29} possible positions may be called an analysis by "brute force", and because it is likely to be the simplest to program it should always be used when the available machine permits it. It is always sound practice to write the simplest program that will do the job in an acceptable time on the equipment available, and this is particularly important when we are seeking to "prove" something by making an assumedly exhaustive search and failing to find a counter-example, because the possibility must always be faced that a machine or program error has caused a valid candidate to be overlooked. However, our assumption here will be that only a small fraction of these 2^{29} positions can be held in immediate-access memory, any further positions having to be held in a "backing store" to which access is much slower. Today, machines offering 2^{29} bits and more of immediate-access memory are commonplace, but it has not always been so, and even with today's machines there are problems for which the immediate-access memory cannot hold all the possible positions. So while what follows will now be academic as regards peg solitaire, peg solitaire provides a convenient medium for its exposition, and it may have relevance to problems which have not yet been attempted.

```
        ·  ·  ·
        ·  1  ·
  ·  ·  ·  ·  ·  ·  ·
  ·  1  ·  1  ·  1  ·
  ·  ·  ·  ·  ·  ·  ·
        ·  1  ·
        ·  ·  ·
```

Figure 3

```
        ·  1  ·
        ·  ·  ·
 –1  1  ·  1  ·  1 –1
  ·  ·  ·  ·  ·  ·  ·
 –1  1  ·  1  ·  1 –1
        ·  ·  ·
        ·  1  ·
```

Figure 4

```
       –1  · –1
        1  ·  1
  ·  ·  ·  ·  ·  ·  ·
  1  ·  1  ·  1  ·  1
  ·  ·  ·  ·  ·  ·  ·
        1  ·  1
       –1  · –1
```

Figure 5

The basic idea is that of *constraints*, which limit the number of positions which need to be stored. These can be divided into *absolute constraints*, which identify positions from which the goal position cannot be reached at all, and *relative constraints*, which identify positions from which it cannot be reached in fewer than a certain number of moves. Three absolute constraints are shown in Figures 3-5. In these, values are attached to the holes of the board, holes indicated by dots having value zero, and the value of a position is obtained by adding up the values of the occupied holes. Figure 3 simply counts the number of key pegs in the position (clearly, we must always retain at least as many as are needed in the goal position). Figure 4 counts the s pegs similarly, the values –1 in the outside corners reflecting the fact that we need an s peg to clear each occupied side corner, and Figure 5 does the same for the t pegs. These are special cases of tables of values for which $f(A) + f(B) \geq f(C)$ for any three holes A, B, and C in line, any such table having the property that the sum of the values of the occupied holes can never increase during the play, but Figures 3-5 show the only tables of this kind that we shall need here.[§]

[‡] This can be proved most simply choosing a direction, say NW-SE, and marking off the diagonals in that direction in threes, *A B C A B C* etc. For any position, we can now count the number of pegs in *A*, *B*, and *C* holes, and note whether $A + B$ and $B + C$ are even or odd (nothing is gained by examining $C + A$ as well, since if $A + B$ and $B + C$ are both even or both odd then $C + A$ is necessarily even, and if one of $A + B$ and $B + C$ are is even and the other is odd then $C + A$ is necessarily odd). This gives us two parity measures, each of which can be even or odd independently, and if we do the same in the NE-SW direction we get two more. Now the effect of a jump is either to leave the count $A + B$ unchanged or to reduce it by 2, so its parity remains unchanged throughout the play, and the same is true of the other counts. So these four parity measures divide the possible positions on the board into sixteen classes, and if the starting position is in a certain class then so must be all the positions derived from it.

[§] In passing, I have to say that I regret the perpetuation of the term "pagoda function" to describe tables of values for which $f(A) + f(B) \geq f(C)$ for any three holes A, B, and C in line. Yes, it is the term that was used by the originators of the idea, which is a cogent and many will think an overwhelming reason for its retention, and it is enjoyably and even gloriously picturesque, but in truth there is nothing remotely pagoda-like about Figures 3-5, and the employment of such a term inevitably gives the impression that something deep and subtle is going on whereas in truth the matter is very simple. In *The Ins and Outs*, I used the term "resource count", which is relatively banal but does at least indicate the sort of thing that is being measured. "Weighted resource count" might have been even better.

```
    A · B                      3/2 · 3/2
    C · D                       · –1 ·
E C C · D D F          3/2 · 1/2 · 1/2 · 3/2
· · · · · · ·           · –1 ·  ·  · –1 ·
G H H · I I J          3/2 · 1/2 · 1/2 · 3/2
    H · I                       · –1 ·
    K · L                      3/2 · 3/2
```

Figure 6 Figure 7

Figures 6 and 7 show examples of relative constraints. Each of the regions *A*...*L* of Figure 6 has the property that the first move involving it must be a move by a peg within it; as long as it is full, no peg in it can be jumped over. So if *n* of these regions are full in a position *P* and only *m* are full in a target position *T*, any sequence of play from *P* to *T* must involve at least *n* – *m* moves. These regions are normally known as *Merson regions* after Robin Merson, who appears to have been the first to draw attention to their usefulness.

Figure 7 has the property that no move, however complicated, can decrease it by more than 1, so if its value is currently *x* and its value for a target position *T* is *y*, we shall need at least *x* – *y* moves to get to *T* (odd halves rounding to the next whole number above). Indeed, we can say more; any move which starts or finishes on an end-of-centre-line hole such as *d*1 must *increase* it by at least 1/2. So, measured by this particular test, such a move not only makes no progress towards the goal, but even moves us away from it.[**]

The constraints embodied in Figures 3-7 do much to prevent the tree of positions from getting too large, but I found I needed further constraints to get the job done on the computer that was available to me in 1984-85. We need 29 bits to represent each position, which means four bytes of computer memory, and while these don't all have to be in immediate-access memory (we can write out sorted subfiles to backing store and then merge them, much as used to be done when performing commercial data processing using magnetic tape in the 1960s), we do have to be able to get them all into the backing store. In 1984-85, I had only two 100Kb discs available to me, each of which could accommodate no more than 25,580 positions (20 subfiles each containing 1,279 positions plus an "all ones" end-of-file indicator), and this was sharply restrictive. I therefore applied additional constraints based on Figure 8 below. This represents the top six holes on the board, *c*1-*e*2, and we observe that the parity of the number of pegs in the three holes marked α can be changed only by a move starting or ending on *c*2 or *d*1; a move into or out of *c*1, or a move alighting on *c*2 in passing but neither starting nor finishing there, has no effect. By considering the various cases at the various board corners and counting the number of moves by s and t pegs necessary to resolve them, I managed to get the position trees down to a size which my machine could handle, but the programming was messy, and I was quietly relieved to read in 2002 that Jean-Charles Meyrignac had confirmed my results using what were presumably brute-force methods without constraints.

```
    c d e     c d e     c d e     c d e     c d e

1   α α ·     ● ● ●     ● ○ ○     ○ ● ○     ○ ● ○
2   α · ·     ○ ● ●     ○ ○ ●     ○ ○ ○     ○ ● ○
```

Figure 8 Figure 9 Figure 10 Figure 11 Figure 12

However, my present purpose is to expound the methods rather than to advance the state of the art, and I decided to use the simpler tests typified by Figures 9-12. Here, we assume a need to clear the end row *c*1-*e*1, and we observe that in the case of Figure 9 we need a delivery of a t peg to either *c*2 or *e*2 (the latter as part of a sequence *e*1-*e*3, *e*4-*e*2, *c*1-*e*1-*e*3). In Figure 10, we need a delivery to either *c*2 or *d*1, and in Figure 11 we need a delivery to *d*2 *unless* the constraint of Figure 7 prohibits a move out of *d*1, in which case we need two deliveries, to *c*2 and to *e*2. Similarly, in the case of Figure 12, if the constraint of Figure 7 prohibits a move out of *d*1 then we again need deliveries to *c*2 and to *e*2.

[**] In *The Ins and Outs*, I attribute the equivalent of Figure 7 to John Conway; in *Winning Ways*, it is attributed to myself. Both attributions are defensible. When, back in 1964 or thereabouts, I sent him my proof that a 17-move solution to the "central game" (start by vacating the central hole, and play to leave a single peg in this same hole) was impossible, I used an argument in words along the lines of "if a move removes more than two inside corner pegs, it must also remove an outside key peg". He immediately send me a diagram equivalent to Figure 7, which encapsulated the matter in numerical terms, and I have used this when expounding the proof ever since. In other words, the encapsulation embodied in Figure 7, which I think valuable, was his, but the observation which it encapsulated was mine.

These deliveries by s, t, and key pegs which are needed to clear *c*1-*e*1 and the corresponding rows elsewhere must be added to the necessary corner-peg moves, and by considering also the absolute constraints of Figures 3-5 and the relative constraints of Figures 6 and 7 we obtain the figures shown in Tables 1 and 2. Table 1 shows the growth and decline of the position tree for the central game (for the moment, ignore the individual columns and consider just the totals on the right), and Table 2 that for the problem "start by vacating *c*1 and play to finish there", which with these constraints is the most demanding computationally of the problems where we aim to finish in the hole which we vacated initially. In Table 1, we consider positions which can be reflected or rotated into each other as the same, so at level 1 we have only one position (as for example after *d*2-*d*4) and not four.

Level	14	15	16	17	18	Total
0	1	0	0	0	0	1
1	0	1	0	0	0	1
2	0	1	1	0	0	2
3	0	1	4	2	2	9
4	0	0	11	20	10	41
5	0	0	12	64	104	180
6	0	0	22	141	406	569
7	0	0	4	324	1219	1547
8	0	0	1	193	2962	3156
9	0	0	0	93	3089	3182
10	0	0	0	24	2530	2554
11	0	0	0	5	1454	1459
12	0	0	0	0	809	809
13	0	0	0	0	269	269
14	0	0	0	0	118	118
15	0	0	0	0	41	41
16	0	0	0	0	8	8
Total	1	3	55	866	13021	13946

Table 1: position counts for a constrained analysis of the central game

Level	11	12	13	14	15	16	Total
0	1	0	0	0	0	0	1
1	0	2	0	0	0	0	2
2	0	4	2	0	0	0	6
3	0	7	22	3	0	0	32
4	0	1	88	99	16	0	204
5	0	0	135	619	477	120	1351
6	0	0	63	1802	3743	2338	7946
7	0	0	51	2042	13732	19971	35796
8	0	0	11	2161	22943	81715	106830
9	0	0	5	1377	29233	159527	190142
10	0	0	0	734	28130	222530	251394
11	0	0	0	498	18005	253325	271828
12	0	0	0	280	9405	203528	213213
13	0	0	0	92	5166	117775	123033
14	0	0	0	0	2019	58327	60346
15	0	0	0	0	0	23803	23803
Total	1	14	377	9707	132869	1142959	1285927

Table 2: the corresponding table for the problem "vacate *c*1, play to finish at *c*1"
(the shortest solution to this problem demands 16 moves)

The columns of Tables 1 and 2 show the moves needed to reach the goal as assessed by Figures 6-7 and the need for deliveries as typified by Figures 9-12; if in Table 2 we are at level 6 and the assessment is that any solution will need at least 8 more moves, that position is counted in column 6 + 8 = 14. And we may note that these tests are merely restrictive; for one of them to say "any solution from this position must take at least *N* more moves" does *not* imply that a solution in *N* moves exists, or even that a solution exists at all.

The utilisation of endgame tables

All this has been a forward search. Let us now look at the effect of adding a search backward from the goal. Specifically, let us borrow a phrase from the field of computer chess, and talk about "endgame tables". If you play chess against a contemporary chess program of any quality, you will find that it has access to precompiled tables which give the result for any position with up to five or even six men (so that once the material on the board has been reduced to this point, the computer can give up the boring business of analysing chess positions, and can go back to downloading pictures of young lady computers with no covers on).

If we compile similar tables for peg solitaire, we find that they expand extremely quickly. In the case of the central game, we see from Table 1 that the successive levels of the forward tree have 1, 2, 9, and 41 members. If we compile an endgame table without constraints, we find that the successive levels have 1, 16, 979, and 14115 members, and even if we apply the constraints of Figures 3-7 (which apply to endgame tables just as to forward searches) these figures are reduced only to 1, 15, 656, and 8993. Even this reduced level-4 number of 8993 is nearly three times as big as the *largest* row total in the forward search. In the case of "vacate *c*1, play to finish at *c*1", we find that the first two levels even of a constrained compilation contain 197 and 5785 members; I don't know how many the third level contains, since I was using the "AVL tree" algorithm and by restricting the tree size to 32767 I could get the two addresses and two one-bit balance indicators neatly into four bytes, but it certainly exceeds 26785. It is surely going to be less than the level-13 total of the forward search tree, but I would not care to back the level-4 total of the endgame table to be less than the level-12 total of the forward tree.

So endgame tables are not going to be a great deal of help, and if they have to be held in immediate-access memory which could otherwise be used to speed up the forward search they may even be disadvantageous. There is however one important respect in which they can contribute. Consider the central game. The goal position has value 0 according to Figure 7, but the last move must gain 1, and the penultimate move can lose at most 1/2. So an appropriate target, if we are aiming for a solution in N moves, is to reduce to value –1/2 at move $N – 2$ rather than to value 0 at move N. Similar remarks apply to the other single-survivor problems.

Can we do better?

The calculations which produced Table 1 did two things: they found a solution in 18 moves, and (subject to the usual provisos concerning the possibility of machine or program error) they proved the non-existence of a solution in 17 moves. To prove the non-existence of a solution in 17 moves required examination of all the positions reported in columns 15-17, and in the absence of more effective constraints I do not see how such a proof could be produced without examining all these positions. However, we might perhaps hope to be able to find an 18-move solution without having to examine all the positions in column 18.

For example, consider Ernest Bergholt's 18-move solution as reported in his 1920 *Complete Handbook to the Game of Solitaire on the English Board of Thirty-three Holes*. In our notation, this goes *d*2-*d*4, *f*3-*d*3, *e*1-*e*3, *e*4-*e*2, *e*6-*e*4 (5), *g*5-*e*5, *d*5-*f*5, *g*3-*g*5-*e*5, *c*3-*e*3, *a*3-*c*3, *b*5-*d*5-*f*5-*f*3-*d*3-*b*3 (11), *c*1-*e*1-*e*3-*e*5, *c*7-*c*5, *c*4-*c*6, *e*7-*c*7-*c*5, *a*5-*a*3-*c*3, *c*2-*c*4-*c*6-*e*6-*e*4-*c*4, *b*4-*d*4 (18), and if we examine the successive positions according to the Merson counts, the need to reduce the value according to Figure 7, and the need for deliveries typified by Figures 9-12, we find that after *d*2-*d*4, *f*3-*d*3, and *e*1-*e*3 we can apparently still hope to find a solution in 15 moves, after *e*4-*e*2, *e*6-*e*4, and *g*5-*e*5 in 16 moves, and after *d*5-*f*5 and *g*3-*g*5-*e*5 in 17 moves. If we repeat Table 1 with these counts to which these positions contribute highlighted in bold, we obtain Table 3 below.

Level	14	15	16	17	18	Total
0	**1**	0	0	0	0	1
1	0	**1**	0	0	0	1
2	0	**1**	1	0	0	2
3	0	**1**	4	2	2	9
4	0	0	**11**	20	10	41
5	0	0	**12**	64	104	180
6	0	0	**22**	141	406	569
7	0	0	4	**324**	1219	1547
8	0	0	1	**193**	2962	3156
9	0	0	0	93	**3089**	3182
10	0	0	0	24	**2530**	2554
11	0	0	0	5	**1454**	1459
12	0	0	0	0	**809**	809
13	0	0	0	0	**269**	269
14	0	0	0	0	**118**	118
15	0	0	0	0	**41**	41
16	0	0	0	0	**8**	8
Total	1	3	55	866	13021	13946

Table 3: Table 1 repeated with the counts to which the Bergholt solution contributes highlighted

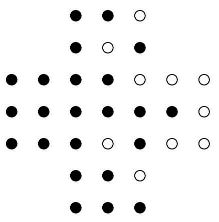
Table 3 suggests that if we rerun the calculation, not only rejecting positions for which the shortest solution will take 19 moves or more but also rejecting positions for which it will take at least two moves more than some other position at the same level, we might obtain an 18-move solution with considerably less effort. This does indeed prove to be the case, as Table 4 overleaf demonstrates.

Level	14	15	16	17	18	Total
0	1	0	0	0	0	1
1	0	1	0	0	0	1
2	0	1	1	0	0	2
3	0	1	4	1	0	6
4	0	0	11	19	0	30
5	0	0	12	63	5	80
6	0	0	22	140	3	165
7	0	0	4	322	11	337
8	0	0	1	192	148	341
9	0	0	0	93	1240	1333
10	0	0	0	24	1476	1500
11	0	0	0	5	951	956
12	0	0	0	0	617	617
13	0	0	0	0	200	200
14	0	0	0	0	101	101
15	0	0	0	0	35	35
16	0	0	0	0	8	8
Total	1	3	55	859	4795	5713

Table 4: Table 1 recalculated with positions apparently non-optimal by at least two moves rejected

Here, we have found an 18-move solution with fewer than half the position examinations needed when compiling Table 1. The reason for the rogue entries such as "1" for "level 3, shortest possible solution 17" is that positions at this level requiring 16 or 17 moves according to the tests turned up before one requiring only 15. Subsequent positions requiring 17 moves were therefore excluded, but it wasn't convenient to go back and turf out any which had already been accepted.

However, Table 3 also suggests that if we rerun the calculation accepting only apparently optimal solutions at each level we might not find an 18-move solution at all, and this does indeed prove to be the case. Examination of the Bergholt solution shows why. After move 8 (*g*3-*g*5-*e*5) we have



and the counts of Figures 6-7 and the deliveries typified by Figures 9-12 do not prohibit a solution in 17 moves, but the only move to maintain this property is *c*1-*e*1-*e*3, and after this move it is in fact impossible to solve the problem even in 18 moves (Bergholt plays *c*3-*e*3 here). This is of course a consequence of the imperfection of our constraints, which merely indicate what is not possible and not what is; but if we had perfect constraints we would already have solved the problem.

The same turns out to be true of all the solvable single-vacancy single-survivor problems. If we reject positions requiring at least two moves more than the best found so far at this level, we obtain a shortest solution with considerably less effort than if we merely reject positions requiring more than the assumed number of moves in the shortest solution, but if we restrict ourselves to apparently optimal moves at each level we do not find a shortest solution at all. Table 5 overleaf shows the result of applying this process to the problem "vacate *c*1, play to finish at *c*1", and if we compare this with Table 2 we find that not only is the number of positions we have to examine in order to obtain a solution in 16 moves only a small fraction of the number previously demanded, but it is even little more than half the number needed to prove the non-existence of a solution in 15 moves.

With hindsight, therefore, we can see that a viable strategy, in the days before machines offering 2^{29} bits of immediate-access memory became readily available, would have been to use a two-stage process: stage A, to find a shortest solution rejecting positions requiring at least two moves more than the best found so far at the current level, and stage B, to prove the non-existence of a shorter one using an examination retaining all possible candidates. If in fact stage A had failed to find the shortest solution, because at some point this solution passed through a position requiring at least two moves more than some other position at that level, it would have turned up during stage B, and no harm would have been done beyond the wasting of a little computing time on an unsuccessful attempt to find a short cut.

Level	11	12	13	14	15	16	Total
0	1	0	0	0	0	0	1
1	0	2	0	0	0	0	2
2	0	4	2	0	0	0	6
3	0	7	22	3	0	0	32
4	0	1	88	71	2	0	162
5	0	0	135	591	11	0	737
6	0	0	63	1755	1	0	1819
7	0	0	51	2030	26	0	2107
8	0	0	11	2153	513	0	2677
9	0	0	5	1375	61	47	1488
10	0	0	0	734	10075	0	10809
11	0	0	0	498	10679	297	11474
12	0	0	0	280	6913	551	7744
13	0	0	0	92	4335	295	4722
14	0	0	0	0	1839	17888	19727
15	0	0	0	0	0	13431	13431
Total	1	14	377	9582	34455	32509	76938

Table 5: Table 2 recalculated with positions apparently non-optimal by at least two moves rejected

Other boards

The performance of the analyses described here depends critically on the constraints used, and these in turn are specific to the standard 33-hole board. Other boards require different constraints, and in particular the 6x6 square board requires little more than a consideration of the 16 Merson regions

$$\begin{array}{cccccc}
A & B & B & C & C & D \\
E & F & F & G & G & H \\
E & F & F & G & G & H \\
I & J & J & K & K & L \\
I & J & J & K & K & L \\
M & N & N & O & O & P
\end{array}$$

These immediately restrict us to 16-move solutions if the initial vacancy is in a corner (because the first move, while broaching one Merson region, will now fill another) and to 15-move solutions otherwise, and it was Robin Merson's observation of this in 1962 that has caused his name to be attached to these regions. John W. Harris had found a 16-move solution to "vacate a1, play to finish there", and Harry O. Davis subsequently found 16-move solutions to the other solvable single-vacancy single-survivor problems with an initial corner vacancy (because the final position must be in the same class as the initial position, a single-vacancy single-survivor problem on this board, as on the standard 33-hole board, is potentially solvable only if the survivor ends in a hole which is a multiple of three holes in each direction away from the initial vacancy). Davis also found 15-move solutions to two problems with the initial vacancy other than in a corner, and Harris subsequently added a third. Harris then attacked the remaining potentially solvable problems by computer, and by August 1986 he had found 15-move solutions to all but one and had proved this one to require 16 moves. I don't know what method he used, but a typical home computer of the period offered a mere 2^{16} bytes of immediate-access memory for program, data, and operating system together, and I imagine that he used techniques somewhat along the lines of those outlined here but with the need to broach Merson regions as his constraint.[††]

[††] I reported all this briefly in the 1992 edition of *The Ins and Outs* and in greater detail in issue 28 (2003) of George Jelliss's *The Games and Puzzles Journal* (see elsewhere on this site), the latter on the grounds that the work in question had been performed seventeen years before, that other people were beginning to reproduce Harris's results, and that I ought to report what I knew if only to establish his priority. However, it appears that this was unnecessary. When I had temporary custody of David Pritchard's chess papers following his death, I found that these included a complete run of Michael Keller's *World Game Review*, and in one of these was an article by Harris which almost certainly contained his own report of the work in question. Unfortunately I neither noted the reference nor made a copy. These copies of *World Game Review* were forwarded to the Musée Suisse du Jeu in June 2012 with the rest of David's chess variant papers (see "The Pritchard archive" under "Chess Variants" elsewhere on this site).

Summary and conclusions

This investigation was sparked off by a wish to explain the different experiences of Barker and Kopf and of myself in respect of the usefulness of endgame tables in calculations of this kind. There are perhaps two reasons for this.

- My objective in 1984-85 was not to find a solution in *N* moves, a task which I judged to be beyond the machine which I had available at the time, but merely to prove the non-existence of a solution in $N-1$ moves. I was therefore not expecting the forward search tree and the endgame table to meet. In fact, rather than have the computer simply report "cannot find", I got it to print out all the positions at the furthest level reached, and checked by hand that none of them would lead to a solution (in most cases it was immediately obvious). I would probably have done the same even had an endgame table been available.

- With the constraints employed here, the forward search does not meet an endgame table of any realistic size until well after it has started contracting, and by that time the bulk of the work has already been done. In these circumstances, any saving in computing time resulting from the use of an endgame table is likely to be marginal, and there seems little point in calculating one beyond the point where it starts to provide a realistic target for the tests based on Figures 6 and 7.

This has answered the question which prompted the investigation, to my own satisfaction at least, and in the process I have come across a method which I had not previously considered: to search by levels, but rejecting positions which appear to be at least two moves worse than the best position so far found at this level. Its application to peg solitaire is now academic, but perhaps there will be other contexts in which it will find a use.

My thanks to Seph Barker and George Bell for some pertinent comments on the first version of this paper, and to Seph Barker for some illuminating and entertaining correspondence throughout.