Proceedings CPAIOR'03

Modelling and Solving English Peg Solitaire

Chris Jefferson, Angela Miguel, Ian Miguel and Armagan Tarim
AI Group, Department of Computer Science
University of York, UK
{caj,angiem,ianm,at}@cs.york.ac.uk

Abstract

Peg Solitaire is a well known puzzle which can prove difficult despite its simple rules. Pegs are arranged on a board such that at least one 'hole' remains. By making draughts-like moves, pegs are gradually removed until no further moves are possible or some goal configuration is achieved. This paper considers the English variant, consisting of a board in a cross shape with 33 holes. Modelling Peg Solitaire via CP or OR techniques presents a considerable challenge and is examined in detail. The merits of the resulting models are discussed and they are compared empirically. The sequential nature of the puzzle naturally conforms to a planning problem, hence we also present an experimental comparison with several leading AI planning systems. Other variants of the puzzle, such as 'fool's Solitaire' are also considered.

Introduction

The origins of Peg Solitaire are uncertain. The game has a long history and has been popular for many years. It is believed that the game was invented by a French nobleman who first played it on the stone tiles of his prison cell in the Bastille. There is some doubt, however, since in 1697 Madame la Princesse de Soubize was pictured in a portrait with the game placed beside her, and also Leibnitz wrote about the game in 1710.

Peg Solitaire is played on a board with a number of holes. In the English version of the game considered here, the board is in the shape of a cross with 33 holes, as presented in Figure 2. Pegs are arranged on the board so that at least one hole remains. By making horizontal or vertical draughts-like moves (see Figure 1), the pegs are gradually removed

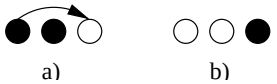

Figure 1: Making a move in Solitaire.

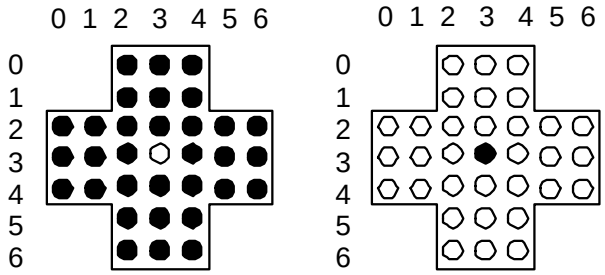

Figure 2: The Solitaire board and first and last states of central Solitaire.

until a goal configuration is obtained. In the classic 'central' Solitaire, (Figure 2), the goal is to reverse the starting position, leaving just a single peg in the central hole.

Beasley [2] surveys the mathematical results known at the time of writing and discusses many closely related puzzles. The major results on Peg Solitaire can be found in Uehara and Iwata [17] on the NP-completeness of the Solitaire problem; De Bruijn [7] on the necessary conditions for feasibility, based on finite field; Berlekamp, Conway and Guy [4] on the necessary conditions for feasibility, using pagoda functions; Avis and Deza [1] on Solitaire cone and the polyhedral approach; and Moore and Eppstein [14] on the one-dimensional problem. Strategies and symmetries are discussed in Beeler et al. [3].

This paper concerns modelling and solving Peg Solitaire using CP and OR techniques. The problem is of interest because it is highly symmetric and because of its sequential nature more usually tackled using AI planning. A CP and an ILP model are developed and combined into a single model most suitable for input into a CP solver. The combined model is shown to be by far the most effective of the three. This model is further enhanced through symmetry breaking and a metric enabling the early detection of dead ends. It is then compared empirically against leading AI planners on a number of variations of the game. Finally, we consider the modifications necessary to our models in order to apply them to an optimisation variant of Solitaire.

Modelling English Peg Solitaire

The possible *transitions* of a peg from one position on the board to another are defined using a coordinate system, as per Figure 2. We adopt the convention that x, y is the xth column and yth row and denote a transition from x, y to x', y' as $x, y \rightarrow x', y'$. We define a *move* as a transition at a particular time-step, t.

One of the key problems in modelling Solitaire is that it involves a sequence of moves. However, it is simple to determine how many moves are necessary:

Lemma 1 *Solitaire requires 31 moves to solve.*

Proof Each move removes exactly one peg. We begin with 32 pegs. Since the objective is to have one peg remaining, 31 moves are required. QED.

Lemma 1 allows the construction of various models with variable(s) for each of the 31 moves, linked by the necessary constraints. Three possibilities are described below.

Model A

Finding a solution to Peg Solitaire may be formulated as an integer programming model. In such a model, the constraints define the feasible moves and purging, and the objective is to have one single remaining peg in the centre of the board. A set of integer variables $M[i,j,t,d]$ is defined to denote the moves to be made from each location $(i,j) \in B$, where B is the set of 33 holes on the board, in each time-step $t = 1, ..., 31$, in any direction $d \in \{N, S, E, W\}$. An additional set of variables, $\mathbf{bState}[i,j,t]$, where $t = 1, ..., 32$, is defined to keep track of the positions with and without pegs on the board.

Constraints ensure that no illegal moves are made. Eqs 2 to 13 ensure that if a peg is selected to jump then it must have an adjacent peg and an empty hole following that adjacent peg. Eqs 14 are transition equations which ensure that the boards in time-steps t and $t+1$ must be consistent with the selected move in time-step t. Eqs 15 imply that there can be one and only one move at each time-step. The objective function, Eq 1, incurs a penalty if the last peg is not in the centre hole in the final stage.

$$\min_{\substack{M\in\{0,1\}\\ \mathbf{bState}\in\{0,1\}}} \sum_{\substack{(i,j)\in B\\ (i,j)\neq \text{centre hole}}} \mathbf{bState}[i,j,32] \tag{1}$$

s.t.

$$M[i,j,t,E] \leq \mathbf{bState}[i,j,t] \tag{2}$$
$$M[i,j,t,E] \leq \mathbf{bState}[i+1,j,t] \tag{3}$$
$$M[i,j,t,E] \leq 1 - \mathbf{bState}[i+2,,t] \tag{4}$$
$$M[i,j,t,W] \leq \mathbf{bState}[i,j,t] \tag{5}$$
$$M[i,j,t,W] \leq \mathbf{bState}[i-1,j,t] \tag{6}$$
$$M[i,j,t,W] \leq 1 - \mathbf{bState}[i-2,j,t] \tag{7}$$
$$M[i,j,t,S] \leq \mathbf{bState}[i,j,t] \tag{8}$$
$$M[i,j,t,S] \leq \mathbf{bState}[i,j+1,t] \tag{9}$$
$$M[i,j,t,S] \leq 1 - \mathbf{bState}[i,j+2,t] \tag{10}$$
$$M[i,j,t,N] \leq \mathbf{bState}[i,j,t] \tag{11}$$
$$M[i,j,t,N] \leq \mathbf{bState}[i,j-1,t] \tag{12}$$
$$M[i,j,t,N] \leq 1 - \mathbf{bState}[i,j-2,t] \tag{13}$$
$$\begin{aligned}\mathbf{bState}[i,j,t] - \mathbf{bState}[i,j,t+1] = &\sum_{d\in\{E,W,S,N\}} M[i,j,t,d]\\ &+ M[i-1,j,t,E] - M[i-2,j,t,E] + M[i+1,j,t,W] - M[i+2,j,t,W]\\ &+ M[i,j-1,t,S] - M[i,j-2,t,S] + M[i,j+1,t,N] - M[i,j+2,t,N]\end{aligned} \tag{14}$$

where $(i,j) \in B, \quad t = 1, ..., 31$ in (2) - (14)

$$\sum_{(i,j)\in B} (M[i,j,t,E] + M[i,j,t,W] + M[i,j,t,S] + M[i,j,t,N]) = 1 \tag{15}$$

where $t = 1, ..., 31$

Model B

Model B uses an array, $\mathbf{moves}[t]$, $t = 1, .., 31$, to record the sequence of moves required to solve the puzzle. The domain of each variable in $\mathbf{moves}[]$ is one of the 76 possible transitions, as enumerated in Table 1. Individual pegs are not distinguished: otherwise, there would be a much greater number of transitions to consider, many of which would be symmetrical since different pegs can be moved to the same coordinate position.

The problem constraints can be stated on $\mathbf{moves}[]$ alone. For example, if transition 0 (i.e. 2,0→4,0) is to be made at move t, then the following preconditions must be satisfied:

1. There must be a peg at 2,0.
2. There must be a peg at 3,0.
3. There must be a hole at 4,0.

No.	Trans.	No.	Trans.	No.	Trans.	No.	Trans.	No.	Trans.
0	2,0→4,0	16	2,2→2,4	32	2,3→0,3	48	1,4→3,4	64	6,4→6,2
1	2,0→2,2	17	2,2→0,2	33	2,3→2,1	49	1,4→1,2	65	6,4→4,4
2	3,0→3,2	18	3,2→3,0	34	2,3→5,2	50	2,4→0,4	66	2,5→4,5
3	4,0→2,0	19	3,2→5,2	35	2,3→4,2	51	2,4→2,2	67	2,5→2,3
4	4,0→4,2	20	3,2→3,4	36	3,3→1,3	52	2,4→4,4	68	3,5→3,3
5	2,1→4,1	21	3,2→1,2	37	3,3→3,1	53	2,4→2,6	69	4,5→2,5
6	2,1→2,3	22	4,2→4,0	38	3,3→3,5	54	3,4→1,4	70	4,5→4,3
7	3,1→3,3	23	4,2→6,2	39	3,3→5,3	55	3,4→3,2	71	2,6→4,6
8	4,1→2,1	24	4,2→4,4	40	4,3→2,3	56	3,4→5,4	72	2,6→2,4
9	4,1→4,3	25	4,2→2,2	41	4,3→4,1	57	3,4→3,6	73	3,6→3,4
10	0,2→0,4	26	5,2→3,2	42	4,3→4,5	58	4,4→2,4	74	4,6→2,6
11	0,2→2,2	27	5,2→5,4	43	4,3→6,3	59	4,4→4,2	75	4,6→4,4
12	1,2→3,2	28	6,2→6,4	44	5,3→3,3	60	4,4→6,4		
13	1,2→1,4	29	6,2→4,2	45	6,3→4,3	61	4,4→4,6		
14	2,2→2,0	30	0,3→2,3	46	0,4→0,2	62	5,4→3,4		
15	2,2→4,2	31	1,3→3,3	47	0,4→2,4	63	5,4→5,2		

Table 1: The set, T, of Solitaire transitions.

These preconditions can be checked by examining the sequence of moves before move t to ensure that the *most recent* moves made involving these three coordinates leave the board in the required state. In general, every transition, τ, involves three positions (and therefore preconditions, $\text{Pre}(\tau)$). Therefore, each move is constrained as follows:

$$\forall t \in \{1,..,31\}\ \forall \tau \in T\ \forall p \in \text{Pre}(\tau)\ \forall \tau^+ \in \text{Support}(\tau,p)\ \forall \tau^- \in \text{Conflict}(\tau,p) :$$
$$\mathbf{moves}[t] = \tau \rightarrow \exists g : \mathbf{moves}[g] = \tau^+ \wedge \neg\exists h : g < h < i \wedge \mathbf{moves}[h] = \tau^-$$

where Support() and Conflict() map onto a set of transitions that support and conflict with precondition p respectively. There is also the special case where the initial state establishes a precondition of τ.

Given 31 moves and 76 transitions for three preconditions, this leads to 7,068 constraints. However, each constraint can have a substantial size. In a format more suitable for input to Solver each constraint looks like:

$$\begin{aligned} \mathbf{moves}[t] = \tau \quad &\rightarrow \\ (\vee\{\mathbf{moves}[t-1] = \tau^+ | \tau^+ \in \text{Support}(\tau,p)\}) \quad &\vee \\ ((\vee\{\mathbf{moves}[t-2] = \tau^+ | \tau^+ \in \text{Support}(\tau,p)\}) \wedge & \\ (\wedge\{\mathbf{moves}[t-1] \neq \tau^- | \tau^- \in \text{Conflict}(\tau,p)\})) \quad &\vee \quad ... \end{aligned}$$

The worst case occurs when $t = 31$. By inspection, the largest size of Support(τ, p) is 4 and the largest size of Conflict(τ, p) is 8 when p requires a peg to be present, and symmetrically 8 and 4 when p requires an empty space. Since the Conflict() set appears more frequently, the right hand side of the above implication constraint can contain up to: $30 \times 4 + \frac{29(29+1)}{2} \times 8 = 3{,}600$ sub-terms.

Model C

Model C combines models A and B to remove some of the problems of both. Given both **bState**[] and **moves**[], the problem constraints are easier to post. A set of constraints between the board states and the moves are also necessary to define the next board

state given the current board state and a transition. One obvious way to post these constraints is to describe how each transition affects each of the board positions. There are 4 mappings of each position from each time-step to the next: it could start empty or full, and end empty or full. The two cases where the state of the position remains the same can be combined, since these are caused by exactly the same set of moves. However the cases where the state of a position changes cannot be combined, since different transition will lead to the position changing from empty to full, and from full to empty.

We generate and post 3 constraints for each position at each time period, giving $1,023 * 3 = 3069$ constraints to specify the problem. For position i, j at time-step t:

1. $(\mathbf{bState}[i,j,t] = \mathbf{bState}[i,j,t+1]) \leftrightarrow (\wedge\{\mathbf{moves}[t] \neq \tau \mid \tau \in \text{Changes}(i,j)\})$
2. $(\mathbf{bState}[i,j,t] = 0 \wedge \mathbf{bState}[i,j,t+1] = 1) \leftrightarrow (\vee\{\mathbf{moves}[t] = \tau \mid \tau \in \text{PegIn}(i,j)\})$
3. $(\mathbf{bState}[i,j,t] = 1 \wedge \mathbf{bState}[i,j,t+1] = 0) \leftrightarrow (\vee\{\mathbf{moves}[t] = \tau \mid \tau \in \text{PegOut}(i,j)\})$

where $\text{Changes}(i,j)$ is the set of transitions which change the state of i, j, and $\text{PegIn}(i,j)$ and $\text{PegOut}(i,j)$ are the sets of transitions which place a peg at and remove a peg from i, j respectively. At most 12 transitions can affect any one position, so the right-hand side of the first constraint type can have a maximum of 12 conjuncts. Similarly, a maximum of 4 and 8 transitions can place a peg at or remove a peg from a particular position.

Pagoda Functions

Conway introduced the term Pagoda for certain functions that one can use to prove that some problems in Peg Solitaire are insoluble and to help in solving certain hard problems [4]. A value is assigned to each board position, so that whenever three positions a, b, c are adjacent in a line, their corresponding values satisfy $a + b >= c$. The *pagoda value* of a particular state for a specific pagoda function is calculated by summing the values the pagoda function takes at each board position where there is a peg. Moves in Solitaire can be defined as taking a line a, b, c of positions and removing pegs from a and b while placing one in c. It follows that as moves are made in the Solitaire game, the pagoda value can never increase. The *pagoda condition* is that the pagoda value for an intermediate position must never be strictly less than that of the final position. If this condition is violated, then it is not possible to extend the current partial solution into a full solution.

There infinitely many pagoda functions. However, for any particular problem some pagoda functions are more useful than others. In an intermediate position it must be possible for the pagoda function to take a smaller value than that of the final state or the pagoda condition will be trivially satisfied throughout. It is beneficial for this to be possible for as many intermediate positions as possible in order to detect dead ends early.

The bulk of the experimental evaluation herein is concerned with Solitaire *reversals*. That is, beginning with a single hole at some position on the board and ending with just a single peg at that same position. Hence, three pagoda functions are considered from [4], as presented in Figure 3. Each is sparsely populated with non-zero positions, increasing the likelihood of breaking the pagoda condition and therefore pruning the search.

In addition, we generated a set of new pagoda functions, one for each position, tailored to maximise the value of the pagoda function in that position and reduce it in the other positions while still conforming to the description of a pagoda function. One method of doing this is to set each position to x^d where d is the *Manhattan distance* [16] from the start/end position of a reversal. The Manhattan distance of two board positions (x_1, y_1)

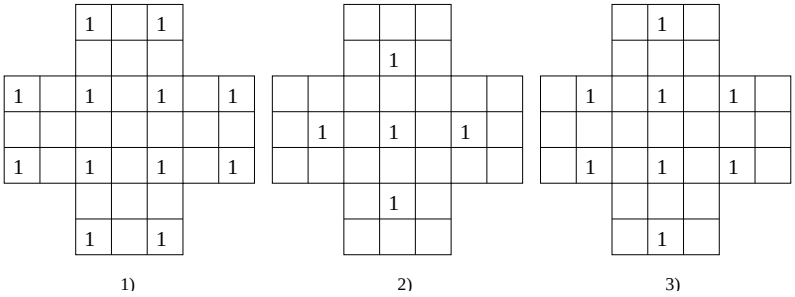

Figure 3: Three Pagoda Functions from [4].

and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$. x must be set in such a way that this is a valid pagoda. Assuming $0 < x < 1$, then trivially $x^2 \leq x + 1$. Hence, the smallest value of x which will satisfy $1 \leq x^2 + x$ is required. Solving this quadratic equation and taking the root with the smallest absolute value gives $\frac{\sqrt{5}-1}{2}$. We will call this the Manhattan pagoda function. An example is presented in Figure 4.

Adding a pagoda function, represented by $pagvals(i, j)$, to models A or C, which have an explicit record of the state of the board, is straightforward. A new array of variables called **pagoda** of length 33 (one per move) is added, and the following are imposed:

1. $\mathbf{pagoda}[t] = \sum_{i,j} pagvals(i, j) \times \mathbf{bState}[i, j, t]$
2. $\mathbf{pagoda}[t] \geq \mathbf{pagoda}[F]$, where F is the final state

Symmetry in Peg Solitaire

Peg Solitaire is highly symmetric, which is a significant factor in the difficulty of the problem. This section discusses these symmetries and methods used to break them.

Board Symmetries

The English Solitaire board has rotational and reflectional symmetry. Consider the central version of the puzzle (Figure 2). Rotational symmetry can be broken by insisting that the first move is transition 7: 3,1→3,3. When finding one solution, however, this is

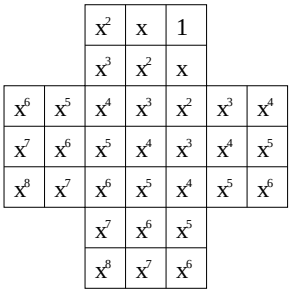

Figure 4: The Manhattan Pagoda Function for (4, 0).

pointless: it is known that a solution exists so the first decision will never be backtracked over. Reflection symmetry persists after the first move, which can be broken by pruning 5,2→3,2 from **moves**[2] in models B and C. This is also unlikely to be useful, since the size of the search space beneath the second decision is substantial. If backtracking to this point is necessary, an unreasonable amount of time is likely to have elapsed.

Further into the search, board symmetries are both broken and subsequently re-established depending on the moves made. Breaking this symmetry is a possible application for SBDS [10] or SBDD [8], and will be investigated further in future work.

Breaking symmetries of independent moves

Many pairs of moves can be performed in any order without affecting the rest of the solution. A move inverts the states of the three pegs upon which it operates, so two moves are *independent* iff the sets of pegs upon which they operate do not intersect. Naive backtracking thrashes on all orderings of each pair of independent moves. To break this symmetry the transitions are ordered (using the ordering in Table 1) and constraints imposed such that adjacent pairs of moves, if independent, must be ordered smallest to largest under this ordering.

Theorem 1 *Given a Boolean function independent*(t, t') *which returns "true" if t and t' are independent transitions, then if a possible solution to Solitaire is represented as an array x of positive integers, giving some fixed order to the possible transitions, then placing the additional constraint independent*$(x[i], x[i+1]) \rightarrow x[i] \leq x[i+1]$ *does not modify the set of unique solutions.*

Proof: Adding constraints cannot increase the set of unique solutions. It remains to show that a solution represented by an array x can always be transformed into a solution represented by an array x' which satisfies the ordering constraints. From the definition of independence, exchanging two adjacent moves $x[i]$ and $x[i+1]$ which do not satisfy the ordering constraints is solution-preserving if $independent(x[i], x[i+1])$. Having made this transformation, and since $x[i+1] < x[i]$, the array x' is now lexicographically less than x. Since both x and x' are finite, a finite number of transformations of this type are possible before reaching an array which satisfies the ordering constraints. QED.

Symmetry breaking constraints of this type can be added straightforwardly to models B and C, which contain the **moves** array. Unfortunately there are other ways in which a sequence of transitions can be interchanged. For example, if transitions 2 and 11 are independent of transition 10 then the above set of ordering constraints would accept both the sequence 10,11,2 and 11,2,10 although they are equivalent. Attempting to break these larger symmetries quickly becomes very complex, both in terms of the number of constraints involved and ensuring that the sets of constraints do not conflict.

Removal of Symmetric Paths

There are often multiple ways of arriving at the same board state. Some of these are due to pairs or sets of moves being interchangeable, as described above. However, others are generated by completely different sets of moves, as presented in Figure 5.

Given a pair of transition sequences like those shown in the figure, the search space beyond the point where they reach the same position is identical. This can lead to a large amount of wasted effort, especially as the number of such symmetrical sequences

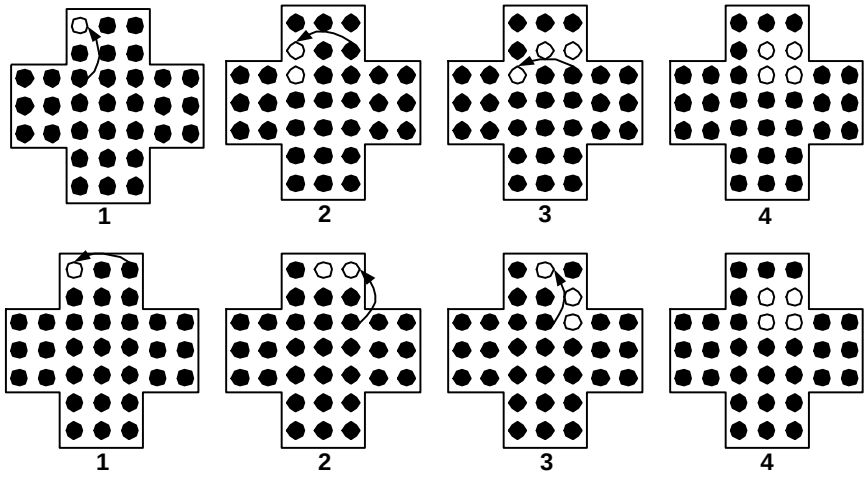

Figure 5: Identical positions reached by symmetric paths.

increases. One way to avoid this problem is to perform a complete search up to some depth, group the transition sequences which lead to identical positions and insert constraints which allow one representative in each group. This approach formed the basis of a small experiment in an effort to reduce the search space substantially in order that more of it can be explored.

Table 2 presents the results of iteratively building up constraints to remove symmetric paths. The experiment was performed with and without pairwise ordering constraints to gauge their utility in removing symmetric paths. At each depth constraints generated at all previous depths are used to minimise the number of extra constraints needed. Even with this incremental approach and the ordering constraints, a substantial set of new constraints is necessary to break the path symmetry by depth 7. Discovering and adding constraints deeper in the search tree is unfortunately not currently feasible. Preliminary experiments also suggest that such a shallow part of the tree is not visited often enough for the overhead of these constraints to be worthwhile.

Without Pairwise Symmetry Breaking				
Depth	Solutions Found	Solutions Pruned	Constraints Added	Time taken (s)
4	400	104	88	<1
5	2228	890	735	1.4
6	11360	5712	4108	16
7	51952	30110	18071	461
Total	221072	199230	23002	
With Pairwise Symmetry Breaking				
Depth	Solutions Found	Solutions Pruned	Constraints Added	Time taken (s)
4	328	32	32	<1
5	1572	234	205	1.5
6	7152	1504	1256	10
7	29953	8111	6167	116
Total	50600	28818	7660	

Table 2: Discovering and adding constraints to remove symmetric paths.

Experimental Results: Solitaire Reversals

All experimentation was performed using a Pentium III 750Mhz and 256 Mb RAM. The constraint and ILP models were solved with Ilog Solver 5.3 and CPLEX 8.0 respectively.

Central Solitaire

We begin by comparing basic versions of models A, B and C on central Solitaire.

The ILP model A fails to return a solution within 12 hours. Several alternative ILP formulations have also been tried without success. Such a failure indicates that this model is intractable from mathematical programming point of view. An examination of the model reveals the probable cause for this failure lies in the objective function. The objective function, which incurs a penalty if the last peg is not in the centre hole, is not a genuine objective in that it can be written simply as a constraint. A direct consequence of having an artificial objective function is the absence of tight bounds, which is a crucial component of any branch-and-bound algorithm used for pruning the search tree.

Given the substantial size of model B, it is not surprising to find that Solver exhausts available memory before all constraints in the model are even posted.

In testing model C, the **moves** array was used to branch on, instantiating the variables in ascending order. The value ordering of the transitions, as given in Table 1, was also ascending. This model, which combines models A and B, proved to be by far the most effective, returning a solution in under 20 seconds. Furthermore, if the final position is unconstrained, a solution is returned in 152 seconds. Clearly some useful propagation is possible from setting the last position.

Comparison with AI Planning Systems

Having established model C as the most effective of the three models developed in this paper, it is further tested on other Solitaire reversals. Model C is very much in the mold of an AI Planning system: the moves array represents the actions of the plan, with preconditions and effects enforced via the **bStates** variables. For this reason, Model C is compared against a variety of leading domain-independent AI planning systems. Blackbox 4.2[1] [11] is a Graphplan-based [5] planner which transforms the planning graph into a large propositional satisfiability (SAT) problem. The solution to this problem, which is equivalent to a valid plan, is obtained by using a dedicated SAT solver. In this set of experiments, the Chaff [15] solver is used. Fast-forward 2.3[2] [12] is a forward-chaining heuristic state space planner which generates heuristics by relaxing the planning problem and solves using a Graphplan-style algorithm. HSP 2.0[3] [6] maps planning instances into state-space search problems that are solved using a variant of the A* search algorithm, with heuristics extracted from the representation. Finally, STAN 4[4] [13] is a Graphplan-based planner which exploits a powerful representational structure for the core planning-graph and a mechanism to reduce search and graph construction costs once the plan-graph structure reaches a stable state.

[1] http://www.cs.washington.edu/homes/kautz/blackbox/
[2] http://www.informatik.uni-freiburg.de/~hoffmann/ff.html
[3] http://www.cs.ucla.edu/~bonet/
[4] http://www.dur.ac.uk/computer.science/research/stanstuff/html/dpgstan.html

```
(:action MOVE-FORWARDS
    :parameters
      (?loc1 ?loc2 ?loc3)
    :precondition
      (and (transition ?loc1 ?loc2 ?loc3)
           (full ?loc1)
           (full ?loc2)
           (empty ?loc3))
    :effect
      (and (empty ?loc1) (not (full ?loc1))
           (empty ?loc2) (not (full ?loc2))
           (full ?loc3) (not (empty ?loc3))))
```

```
(:action MOVE-BACKWARDS
    :parameters
      (?loc1 ?loc2 ?loc3)
    :precondition
      (and (transition ?loc1 ?loc2 ?loc3)
           (empty ?loc1)
           (full ?loc2)
           (full ?loc3))
    :effect
      (and (full ?loc1) (not (empty ?loc1))
           (empty ?loc2) (not (full ?loc2))
           (empty ?loc3) (not (full ?loc3))))
```

Figure 6: STRIPS operators for Solitaire.

Encoding

A simple STRIPS [9] encoding is sufficient to describe Peg Solitaire. The plan objects are the 33 board positions which may be full or empty. Both empty and full predicates are required since some STRIPS planners do not support negated preconditions or goals. Transitions are stated in terms of the three board positions which they involve, e.g. (*transition loc1*, *loc2 loc3*). Move operators can use these transitions in either direction, depending on the state of the relevant board positions, as presented in Figure 6.

Results

The experiments concerned the full set of single-peg Solitaire reversals. From [4] it is known that all possible single-peg reversals are solvable. We set each of our systems the task of solving all such reversals. This may seem unnecessary — after all if a reversal for position (3,0) can be solved, the reversals for (6,3), (3, 6) and (0,3) can be obtained easily by symmetry. However, these symmetrical problems are not identical given our simplistic value ordering (trying each transition in ascending order), nor to the planners which are also given the transitions in a particular order. Hence, the necessity to experiment with the full set for a fair comparison.

The performance of the planning systems is presented in Table 3. The most successful systems, Blackbox and FF, achieved a high percentage of coverage in terms of number of problems solved. The less successful systems appeared to suffer as much from lack of resources as lack of time. The most difficult reversals for the planners to solve were the central reversal (which is one of the reversals solved most easily by the CP approach) and reversals at positions 2 and 3 spaces distant from the centre in a straight line. In particular, the reversal 3 spaces distant from the centre, namely at (3, 0) and symmetric equivalents, is well known as the 'notorious' reversal because of its difficulty [4].

Table 4 (Appendix A) summarises the performance of variants of model C. Although the coverage of this model cannot match that of the best dedicated planning systems, probably due to the naive value ordering, the results demonstrate that model C is effective for a number of reversals besides central Solitaire. It is also compact, never suffering from the memory problems encountered by all the planners. The pairwise symmetry-breaking constraints described above are very effective in reducing search effort, allowing some problems to be solved not previously solvable within one hour. The use of Pagoda functions to guide the search also brings a significant benefit and can be successfully

Rev.	Bbox4.2	FF2.3	HSP2.0	Stan4	Rev.	Bbox4.2	FF2.3	HSP2.0	Stan4
(2,0)	13	49	-	-	(3,0)	-	-	148	-
(4,0)	14	622	-	-	(2,1)	25	121	-	>1hr
(3,1)	543	-	-	-	(4,1)	17	1	-	>1hr
(0,2)	19	0.2	30	-	(1,2)	25	0.7	-	1126
(2,2)	47	1	-	-	(3,2)	48	3544	86	-
(4,2)	38	0.6	27	-	(5,2)	14	273	48	>1hr
(6,2)	21	0.6	32	-	(0,3)	-	-	-	-
(1,3)	862	>1hr	-	>1hr	(2,3)	28	0.1	125	-
(3,3)	-	-	57	-	(4,3)	30	48	-	-
(5,3)	620	>1hr	574	>1hr	(6,3)	-	>1hr	-	-
(0,4)	19	276	-	-	(1,4)	14	0.05	97	>1hr
(2,4)	42	0.15	-	-	(3,4)	44	0.05	313	-
(4,4)	49	0.05	60	-	(5,4)	16	1564	-	>1hr
(6,4)	19	-	125	-	(2,5)	16	553	-	>1hr
(3,5)	-	-	-	>1hr	(4,5)	27	1521	-	>1hr
(2,6)	18	>1hr	298	-	(3,6)	-	-	-	-
(4,6)	21	9.8	154	-					

Table 3: Results in seconds. Dash indicates out of memory.

combined with the symmetry-breaking constraints.

Finally, to test how naive ascending value ordering inhibits the search, a dynamic heuristic was tested (see the 'corner bias' column of Table 4). Transitions which move corner pegs towards the middle (e.g. a transition from (2, 0) to (2, 2)) are preferred. If no such transition is possible, Solver reverts to the naive ascending order. On the occasions that this heuristic is successful a sizeable reduction in search effort is evident, indicating that a more informed heuristic might give a more 'robust' model overall.

Fool's Solitaire

An optimisation variation of Solitaire (named Fool's Solitaire by Berlekamp, Conway and Guy [4]) is to attempt to reach a position where no further moves are possible in the shortest sequence of moves. This section describes methods to solve this problem.

A CP Model

Model C is used with a few small modifications. First, a 77th transition is added, which will be denoted by the keyword 'DeadEnd', representing the fact that the search has reached such a dead-end. Of course, if the search is at a dead-end at time-step t, then it must be in a dead-end at time-step $t+1$, hence the following constraints:

$$\forall t \in \{1, \ldots 30\} : \mathbf{moves}[t] = \text{DeadEnd} \rightarrow \mathbf{moves}[t+1] = \text{DeadEnd}$$

The DeadEnd transition must only be allowed if no other move is possible. A simple way to achieve this is to use an implication constraint based on the **bState**[]. If any of the

preconditions hold for any other transition, the DeadEnd transition is not allowed:

$$\begin{aligned} &\forall t \in \{1,\ldots,31\}: \\ (\mathbf{bState}[2,0,t] = 1 \wedge \mathbf{bState}[3,0,t] = 1 \wedge \mathbf{bState}[4,0,t] = 0) \quad &\vee \ldots \\ \vee(\mathbf{bState}[4,6,t] = 1 \wedge \mathbf{bState}[4,5,t] = 1 \wedge \mathbf{bState}[4,4,t] = 0) & \\ \rightarrow \mathbf{moves}[t] \neq \text{DeadEnd} & \end{aligned}$$

This large arity constraint does not seem to degrade the performance of Solver. Finally, some implied constraints are added: if there is any dead-end solution at all, **moves**[31] must be assigned DeadEnd, and **moves**[0] cannot be DeadEnd by inspection. The objective is to maximise the number of occurrences of the DeadEnd transition in **moves**[].

An IP Model

IP model A is extended to determine the minimum number of moves required to reach a dead-end. A binary decision variable, $C[i,j,t]$, is introduced which equals 1 iff there is a peg in hole (i,j) with a legal move. The objective is to minimise the total penalty incurred by having legal moves. The objective function coefficient 33^{t-1} guarantees that the optimal solution consists of a minimum number of moves leading to a dead-end.

$$\min_{\substack{M\in\{0,1\}\\ \mathbf{bState}\in\{0,1\}\\ C\in\{0,1\}}} \sum_{(i,j)\in B}\sum_{t=1}^{31} 33^{t-1} * C[i,j,t] \tag{16}$$

s.t.

$$M[i,j,t,E] \leq \mathbf{bState}[i,j,t] \tag{17}$$
$$M[i,j,t,E] \leq \mathbf{bState}[i+1,j,t] \tag{18}$$
$$M[i,j,t,E] \leq 1 - \mathbf{bState}[i+2,j,t] \tag{19}$$
$$M[i,j,t,W] \leq \mathbf{bState}[i,j,t] \tag{20}$$
$$M[i,j,t,W] \leq \mathbf{bState}[i-1,j,t] \tag{21}$$
$$M[i,j,t,W] \leq 1 - \mathbf{bState}[i-2,j,t] \tag{22}$$
$$M[i,j,t,S] \leq \mathbf{bState}[i,j,t] \tag{23}$$
$$M[i,j,t,S] \leq \mathbf{bState}[i,j+1,t] \tag{24}$$
$$M[i,j,t,S] \leq 1 - \mathbf{bState}[i,j+2,t] \tag{25}$$
$$M[i,j,t,N] \leq \mathbf{bState}[i,j,t] \tag{26}$$
$$M[i,j,t,N] \leq \mathbf{bState}[i,j-1,t] \tag{27}$$
$$M[i,j,t,N] \leq 1 - \mathbf{bState}[i,j-2,t] \tag{28}$$
$$C[i,j,t] \geq \mathbf{bState}[i,j,t] + \mathbf{bState}[i+1,j,t] - \mathbf{bState}[i+2,j,t] - 1 \tag{29}$$
$$C[i,j,t] \geq \mathbf{bState}[i,j,t] + \mathbf{bState}[i-1,j,t] - \mathbf{bState}[i-2,j,t] - 1 \tag{30}$$
$$C[i,j,t] \geq \mathbf{bState}[i,j,t] + \mathbf{bState}[i,j+1,t] - \mathbf{bState}[i,j+2,t] - 1 \tag{31}$$
$$C[i,j,t] \geq \mathbf{bState}[i,j,t] + \mathbf{bState}[i,j-1,t] - \mathbf{bState}[i,j-2,t] - 1 \tag{32}$$

$$\begin{aligned} \mathbf{bState}[i,j,t] - \mathbf{bState}[i,j,t+1] = &\sum_{d\in\{E,W,S,N\}} M[i,j,t,d] \\ &+ M[i-1,j,t,E] - M[i-2,j,t,E] + M[i+1,j,t,W] - M[i+2,j,t,W] \\ &+ M[i,j-1,t,S] - M[i,j-2,t,S] + M[i,j+1,t,N] - M[i,j+2,t,N] \end{aligned} \tag{33}$$

where $(i,j) \in B, \quad t = 1,...,31$ in (17) - (33)

$$\sum_{(i,j)\in B} (M[i,j,t,E] + M[i,j,t,W] + M[i,j,t,S] + M[i,j,t,N]) \leq 1 \tag{34}$$

where $t = 1,...,31$

The new constraints, Eqs. 29–32, set $C[i,j,t] = 1$ if there is a legal move in time-step t, and hence it is possible to charge a penalty for each legal move.

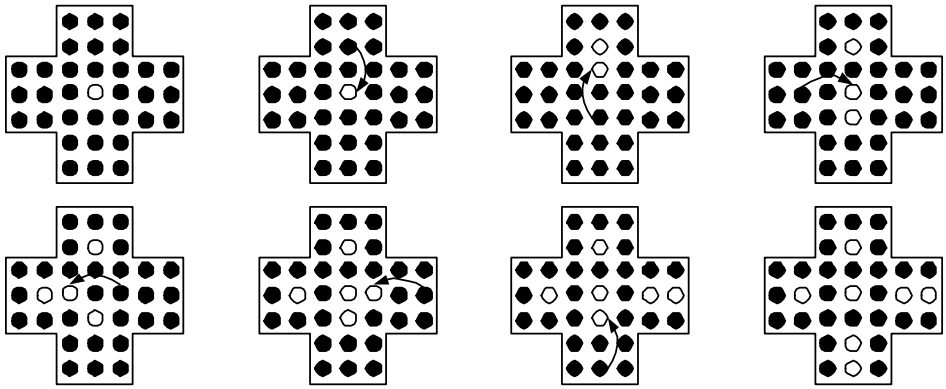

Figure 7: An optimal route to a dead-end in central Solitaire.

Results

For the CP model, if the same search strategy is used as for the Solitaire reversals, namely instantiating **moves**[] from 1 to 31 and assigning the transitions in lexicographic order, this model does very poorly. This is because the original search strategy seems to be very well suited to finding a solution - the opposite of what is now required. If, however, **moves**[] is instantiated in reverse order and the transitions are assigned in reverse lexicographic order the model can be used to prove that 6 moves is the optimal in 20 seconds. The sequence of moves to reach the dead-end is presented in Figure 7. This strategy is successful because it tries to force as many as possible of the elements of **moves**[] to be the DeadEnd transition as early as possible. This provides a much better bound on the number of occurrences of DeadEnd.

The IP model is solved in 27 seconds. The IP formulation given above is intractable due to the large number of decision variables. In order to reduce the number of decision variables an iterative procedure is adopted and a new variable is defined only when it is needed. Initially, only two moves are allowed. The number of allowed moves is increased by one in each iteration until the dead-end in Figure 7 is observed.

Conclusions and Future Work

This paper has concerned the modelling and solving of certain variations of English Peg Solitaire. It was shown how basic, and ineffective, CP and ILP models could be combined into a superior single model which was used to solve the classic central Solitaire puzzle very efficiently. This is another instance of the efficacy of channelling between two complementary models: here one model supplies the constraint that there can be only one move at once, and usefully groups three state changes into a single number. The other allows us to state preconditions on each possible transition, without considering the entire move history.

In experimental tests the combined model, enhanced with symmetry-breaking constraints and a specialised function used to detect dead ends, bore comparison with dedicated AI planning systems. With minor modifications it was also used to solve an

optimisation variant of the problem for which no simple AI planning model exists.

The lessons learned from modelling Solitaire should generalise to other sequential 'planning-style' problems. Using channelling constraints to specify action pre- and post-conditions and the techniques used to break symmetries of independent moves and to remove symmetrical paths should prove particularly useful. In order to improve the combined model further, we will investigate the heuristic *packaging* system suggested in [4]. We will also examine further configurations of English Solitaire as well as the more difficult French variation. Alternative optimisation variants will also be investigated, such as optimising the number of draughts-like multiple moves using the same peg.

Acknowledgements. This research is supported by UK EPSRC grant numbers GR/N16129 and GR/R30792. We thank Chris Beck, Alan Frisch, Steve Linton and members of the APES research group, especially Barbara Smith and Toby Walsh for their insightful comments.

References

[1] D. Avis and A Deza. *On the Boolean Solitaire Cone*. Technical Report, McGill University and Tokyo Institute of Technology, 1999.

[2] J. D. Beasley. *The Ins and Outs of Peg Solitaire*. Oxford University Press, Oxford, 1992.

[3] M. Beeler, R. W. Gosper and R. Schroeppel. *HAKMEM*. Technical Report, MIT, Artificial Intelligence Laboratory, Memo AIM-239, 1972.

[4] E.R. Berlekamp, J.H. Conway, R.K. Guy. *Winning Ways for your mathematical plays, Vol.2: Games in Particular*. Academic Press, London, pp. 729-730, 1982.

[5] A. Blum and M. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90, pp:281-300, 1997.

[6] B. Bonet and H. Geffner. Planning as Heuristic Search. *Artificial Intelligence* 129, pp. 5-33, 2001.

[7] N. G. de Bruijn. A Solitaire Game and its Relation to a Finite Field. *Journal of Recreational Mathematics*, 5, pp: 133-137, 1972.

[8] T. Fahle, S. Schamberger and M. Sellman. Symmetry Breaking. *Proceedings Principles and Practice of Constraint Programming*, LNCS 2239, pp 93–107, 2001.

[9] R. Fikes and N. Nilsson. Strips: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 5(2), pp:189-208, 1971.

[10] I. Gent and B. Smith. Symmetry Breaking in Constraint Programming. *Proceedings European Conference on AI '2000*, pp 599–603, 2000.

[11] H. Kautz and B. Selman. Unifying SAT-based and Graph-based Planning. *Proceedings International Joint Conference on Artificial Intelligence*, pp. 318–325, 1999.

[12] J. Koffman and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14, pp. 253-302, 2001.

[13] D. Long and M. Fox. Efficient Implementation of the Plan Graph in STAN. *Journal of Artificial Intelligence Research* 10, pp. 87-115, 1999.

[14] C. Moore and D. Eppstein. One-dimensional Peg Solitaire, and Duotaire. In, R.J. Nowalski (ed), *More Games of No Chance*, Cambridge University Press,pp. 341–350, 2000.

[15] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik. Chaff: Engineering an Efficient SAT Solver *Proceedings 38th Design Automation Conference (DAC'01)* 2001.

[16] S. Skiena. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Addison-Wesley, 1990.

[17] R. Uehara and S. Iwata. Generalized Hi-Q is NP-Complete. *Trans IEICE*, 73, pp:270-273, 1990.

Appendix A: CP Results for Solitaire Reversals

	Lexicographic Value Ordering														
	Basic Model			Pairwise Sym.Break.			Pagoda Functions			Pagoda + Sym.Break.			Corner Bias		
	time	fails	choices	time	fails	choices	time	fails	choices	time	fails	choices	time	fails	choices
(3,1)	2903	460393	460387	221.5	15527	15551	2730.7	340268	340292	54.6	4486	4510			
(0,2)	439	46399	46423	61.1	4967	4991	443	46399	46423	64.9	4967	4991			
(4,2)	7	628	654	2.7	147	173	8	628	654	2.7	147	173	0.7	2	30
(2,3)	17	1573	1599	3.5	322	348	18	1573	1599	4.9	322	348	1.2	38	64
(3,3)	16	1372	1398	4.1	391	415	7.9	926	951	5.0	391	415	1.3	50	77
(4,3)	1700	264385	264412	197.0	13758	13785	1712	264385	264412	199.9	13758	13785			
(1,4)				1891.2	130868	130686				1036.0	64879	64905			
(2,4)	116	9023	9051	19.1	1287	1315	102	8960	9001	22.3	1287	1315	1.9	108	135
(3,4)				337.8	24217	24243				349.6	24217	24243			
(4,4)													0.7	0	28
(2,5)													1.2	73	45
(4,6)													5.5	420	446

Table 4: Solitaire reversals solved via CP models. Only those reversals with at least one positive result are shown. A blank indicates no solution within 1 hour.

Use of Pagoda Functions:
Pagoda function 1) $\{(0,2),(4,2),(2,4),(4,4),(2,6)\}$
Pagoda function 2) $\{(3,1),(3,3)\}$
Pagoda function 3) $\{(1,4),(3,4)\}$
Pagoda function 3) rotated $\{(2,3),(2,5),(4,3)\}$
The Manhattan Pagoda functions were not found to provide any extra benefits.