Informed Search Using Equivalent-Class Templates*

Dragoş-Anton Manolescu Todd E. Morgan
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{manolesc,temorgan}@cs.uiuc.edu

Abstract

Many of the search techniques used in AI were developed, studied and/or optimized in the context of game playing. Game playing demands efficient methods and has the advantage that the testing and evaluation of different implementations is straightforward, for it requires playing the game. This paper provides a case study of an informed search technique applied for a non-adversarial game (triangular board Peg Game). The performance of the method is compared with other strategies that were tried. In addition, conclusions about the peg game are presented.

1 Introduction

One of the AI techniques used extensively in computer game playing is search. Although human game-playing relies on pattern-based strategies, most computer game-playing systems do not. Instead, from any given state these systems typically generate a tree of possible moves from which the best possible move is chosen. Exhaustive search will yield an answer, although as the size of a search tree is often-times large, the amount of computation required is unacceptable. One other solution is to use a heuristic search that explores only those regions of interest in the search tree.

The rest of this paper presents a solution to the Peg Game. The solution uses a combination of two strategies—pattern-based search and heuristic search. The first section of the paper introduces the game and describes the problem statement. The second section covers some search techniques used in AI programs that deal with games. The third section presents different search strategies that were tried and ends with a detailed overview of the proposed combined approach. The paper concludes with a summary and explains how this strategy could be used to solve other problems.

1.1 Problem description

Although the Peg Game has several different variants, they are all similar with respect to the way the board is set up and how they evolve towards the end of the game. The board has a number of holes which can be either empty or occupied by pegs. In order to move a peg, it has to jump (in straight line) over an occupied hole (i.e. another peg) and be inserted into a free hole. The peg from the middle hole is then removed from the board. Therefore, each move removes one peg from the board. The game starts with exactly one hole empty at random, and the goal is to remove all pegs but one. This is not an easy task, considering the constraints imposed by the board geometry as well as the choice of valid moves.

Figure 1 depicts the board for the Peg Game (equilateral triangle with 5 holes on each side, 15 holes total), some possible moves from a given state, and the labeling convention. This particular geometry implies several interesting properties (some of which appeared evident during experiments with various search techniques):

- The game ends at most after 14 moves when there is only one peg on the board, or before that when there are no other possible moves; the former corresponds

*Proceedings of the 5th ISCA Conference (pages 197–201), edited by F. C. Harris, Jr. June 1996, Reno, Nevada, USA.

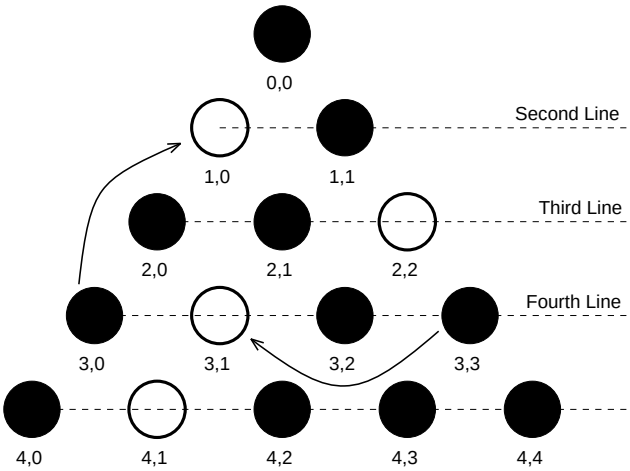

Figure 1: Peg game board

Starting position	Equivalent starting positions
(0,0)	(4,0) (4,4)
(1,0)	(1,1) (3,0) (4,1) (3,3) (4,3)
(2,0)	(2,2) (4,2)
(2,1)	(3,1) (3,2)

Table 1: Equivalence by symmetry

to a winning game and the latter to a stalled game. Therefore, the depth of the game tree is at most 14.

- The number of possible moves at each step is not constant. This can be observed in Figure 2 which shows how the number of possible moves evolves during several winning games. Therefore, the game tree has a variable branching factor with a maximum degree 9.
- There is a high degree of symmetry. Based on this, the total of 15 different initial situations can be reduced to just 4, as shown in Table 1.
- A search in the game tree could generate a succession of 14 moves that would leave just one peg on the board. The search can be either an exhaustive one (i.e. search blindly until a solution is found) or an optimized search that minimizes the search space according to some heuristic. The key point here is to find the right heuristic that will efficiently guide the search.

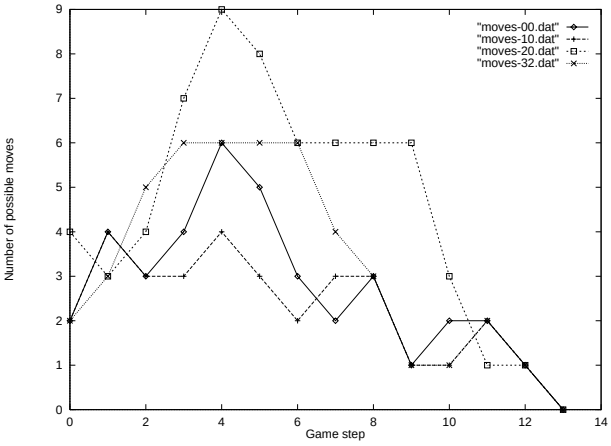

Figure 2: Variance of branching factor during the Peg Game

1.2 Other Approaches

Some of the better known games which have been attempted using AI techniques include Connect-Four, Draughts, Othello, and Backgammon [3]. Each of these

games is most-suited to a particular approach, some of which share several characteristics with our own domain, the Peg Game. Others have been "solved" with techniques that could perhaps apply also to the Peg Game, but remain for the moment a subject of future research (i.e. neural networks and genetic algorithms). Among those similar in approach to the Peg Game are Connect-Four and Draughts.

A Connect-Four playing program called VICTOR [5], relied on a set of strategic rules which were developed by the authors after extensive experimentation with the game. These rules defined optimal moves in any given situation, and could be combined in pre-determined groups. With this method, they were able to create an approach which did not break the game into distinct phases, such as an opening-move or an end-game. The Peg Game lends itself very well to this type of approach.

The other game we found with a solution similar to ours was Draughts. Smeets and Putter [6] developed a technique called the Signature Method, where evaluation functions mapped board states onto very limited ranges (typically $[-1, 1]$), which were combined in multidimensional lookup tables to determine the value of a given move. These tables also allowed the program to learn from previous games. The domain of the Peg Game is simple enough that it was unnecessary to incorporate learning into our solution, but the idea of limited-range evaluation functions coordinated well with that of templates.

The two techniques mentioned earlier as possibilities for future Peg Game research, neural networks and genetic algorithms, have been successfully applied in backgammon [7] and Othello [8]. Both these programs performed very well in their classes, indicating the potential for applying novel techniques in traditional AI game-playing domains.

2 Early approaches

The basic technique used for the Peg Game is hill-climbing—for a detailed description, see [1] and [2]. It relies on a heuristic function that ranks different game states such that the ones which are closer to the solution get explored first. It is clear that the efficiency of the search depends on how well the heuristic function evaluates different board states. The rest of this section describes four different heuristics that we have tried for the Peg Game problem.

Approach 1: The first heuristic presented relies on the observation that the pegs have to be clustered together and not spread on the board, for moves are only possible as long as there is an adjacent peg to jump over. A board state is characterized by a number which is computed in the following way:

```
HEURISTIC 1:
 score=0
 foreach peg on the board
  score=score+number of adjacent pegs
 endfor
```

Therefore, states that keep the pegs together are ranked before the ones where the pegs are scattered on the board.

Approach 2: A second heuristic is obtained by combining the previous one with properties derived from the board shape. From Figure 1 it is clear that there are just 2 possible moves from each corner of the board. Avoiding these positions could cut off some branches in the search tree where the game will stall, and therefore the size of the tree decreases. The board states in which the pegs are clustered together and away from the corners receive high scores.

```
HEURISTIC 2:
 score=0
 foreach peg on the board
  score=score+number of adjacent pegs
 endfor
 foreach corner of the board
  if corner is free then
   score=score+1
  endif
 endfor
```

Approach 3: A third heuristic, which again uses the idea of clustering, directs the pegs to one side of the board. For each side, a weighted sum that takes into account the number of pegs on the side as well as the

pegs on the adjacent line is computed and the maximum of the 3 values is returned as the board's score.

```
HEURISTIC 3:
 score1=0
 score2=0
 score3=0
 foreach side of the board
  scorei=w1*number of pegs on the side+
         w2*number of pegs on the adjacent line
 endfor
 max(score1,score2,score3)
```

Experimenting with these functions, the properties discussed in section 1.1 became evident, as well as the following results:

- Starting from any position on the board, there are at least two different solutions
- The last peg on the board of a complete solution is in the middle of one side

Approach 4: In response to the dependence between performance (of the previous functions) and the starting position, the fourth heuristic implements separate functions optimized for the equivalent classes from Table 1. Each board state is assigned a score that is computed as the sum of key positions.

```
HEURISTIC 4:
 case start
  eq(0,0) : return sum keypos(0,0)
  eq(1,0) : return sum keypos(1,0)
  eq(2,0) : return sum keypos(2,0)
  else    : return sum keypos(2,1)
 endcase
```

In the previous description, `eq(i,j)` is true whenever `start` is a member of the equivalence class (i, j)—see Table 1—and `sum keypos(i,j)` returns the sum of the pegs from the key positions corresponding to the equivalence class (i, j).

Table 2 gives the performance of these heuristics in terms of number of nodes visited in the game tree. Although board symmetry reduces the number of starting positions to just four, the functions perform differently. All the starting positions from the same equivalence class are listed together. This behaviour is a consequence of the fact that the possible board states are generated in a deterministic manner.

Starting position	Heuristic			
	1	2	3	4
0,0	6633	6636	352	14
4,0	6633	6636	6794	20
4,4	6461	6464	6807	185
1,0	6636	6636	6794	195
1,1	6461	6464	6794	24515
3,0	6633	6636	562	352
4,1	6456	6456	6807	24526
3,3	6624	6624	354	673
4,3	6456	5467	562	24522
2,0	85	24	15	46
2,2	85	19	15	17079
4,2	907	30	91	2557
2,1	2803	2803	7656	596
3,1	2818	2818	7704	425
3,2	2818	2820	7670	360

Table 2: Early approaches: Performance in terms of number of nodes explored

3 Final approach

As can be seen in Table 2, certain starting positions resulted in a much more focused, and therefore smaller, search. Starting from the middle position of any side, we were able to find solutions within 100 nodes, as opposed to over 6000 for the other cases. This led us to investigate other methods of guiding the search.

The final approach we used takes into account the symmetry of the board, allowing us to use the principle of "divide and conquer" to split the original problem into four smaller problems, one for each of the equivalence classes (e.g. the symmetric starting positions). By treating each of these cases separately, we created customized heuristics.

The equivalence-class templates specify board patterns that the search should follow in the search tree. There is a template for the first 11 of the 14 game steps. A board state is compared against the corresponding template and if it matches, the heuristic function returns a positive value. Otherwise, it means that an unproductive branch has been reached and the function returns zero. The idea behind the templates is that they do not give lockstep sequences of moves; rather, they indicate a general direction in which the search should continue. In this way the templates guide the search towards a solution, therefore decreasing the number of nodes explored.

Each template is a set of 11 assertions; there are no assertions after step 11 as there are just 3 pegs left, and the search space is so small that exhaustive search suffices. For each step of the game, these assertions must hold in order to consider that state for further expansion. An example is given below[1]:

```
 1. A position from the middle of a side is
    free
 2. Three positions of the same side are free
 3. Two positions in the center are free
 4. A second line is free
 5. The middle of a side is free
 6. A third line is free
 7. A fourth line is free
 8. Four positions are free on a side
 9. Two positions are free on the side
    opposite to the starting position
10. Three positions are free on the side
    opposite to the starting position
11. The three positions in the center are
    free
```

The equivalence-class templates method outperforms the methods discussed previously for three of the four equivalence classes. It appears that this method does not achieve the same performance as the others when the game starts from one of the three middle positions—(2,1), (3,1) and (3,2). A possible explanation for this behaviour is that we are not using a template general enough to cover all these situations. However, the overall performance of the method is better—see Table 3.

[1]The actual templates are implemented in Common Lisp.

Starting position	Template heuristic
0,0	23
4,0	120
4,4	125
1,0	82
1,1	93
3,0	279
4,1	333
3,3	322
4,3	322
2,0	532
2,2	571
4,2	1060
2,1	1027
3,1	89
3,2	31

Table 3: Template heuristic: Performance in terms of number of nodes explored

4 Conclusion

This paper presents an efficient solution to the Peg Game. The search method presented in this work combines ideas from two different paradigms. One is heuristic search, a widely used search technique that is normally applied when a heuristic function that ranks different nodes in the search tree is available. The other, pattern-based strategy, is the foundation of human game-playing and so far there are still areas where it out-scores any machine opponent—for example, the game of Go.

The main characteristics of the equivalence-class templates method are summarized as follows:

- It is an informed search; the knowledge about the problem is encoded in the templates
- It efficiently exploits the representation; due to symmetry there are just 4 distinct cases instead of 15
- It is general; the templates do not give an algorithm that produces the solution, but guide a search in the game tree

- It is flexible; it does not depend on the search algorithm used and can be integrated in different algorithms

- It exhibits better overall performance; "classical" heuristic-guided search [4] does not sufficiently reduce the search space

Performance improvements are possible through further refinement of the actual templates. Each of the templates needs to capture the essential transformations from state to state and be kept general enough that it works for all cases in an equivalence class.

It is our belief that this kind of search strategy could potentially be of great use for other applications which operate in domains containing certain properties, such as a high degree of symmetry. By enabling the computer to combine pattern matching with its ability of high speed search, complex domains can be reduced to manageable size. Non-conventional strategies like genetic algorithms or neural networks have proven their efficiency in domains where traditional methods failed to perform adequately. The proposed method is a good example of combining established techniques with novel approaches.

Acknowledgements

We would like to thank Caroline Hayes for her advice and suggestions and Duncan Lawrie for helping us obtain support from the Department of Computer Science.

References

[1] Patrick Henry Winston, *"Artificial Intelligence, Third Edition,"* Addison-Wesley 1992.

[2] Stuart Russel and Peter Norvig, *"Artificial Intelligence, A Modern Approach,"* Prentice Hall 1995.

[3] D. N. L. Levy and D. F. Beal, *"Heuristic Programming in Artificial Intelligence,"* John Wiley & Sons 1989.

[4] L. Kanal and V. Kumar, *"Search in Artificial Intelligence,"* Springer-Verlag 1988.

[5] J. W. H. M. Uiterwijk, H. J. van den Herik and L. V. Allis, *"A Knowledge-Based Approach to Connect-Four,"* 1st London Conference on Computer Games 1989.

[6] John J. Smeets and Gerard Putter, *"Some Experience with a Self-Learning Computer Program for Playing Draughts,"* 1st London Conference on Computer Games 1989.

[7] Gerald Tesauro, *"Neurogammon: A Neural-Network Backgammon Learning Program,"* 1st London Conference on Computer Games 1989.

[8] Greg M. Gupton, *"Genetic Learning Algorithm Applied to the Game of Othello,"* 1st London Conference on Computer Games 1989.

[9] Patrick Henry Winston and Berthold Klaus Paul Horn, *"Lisp, 3rd Edition,"* Addison-Wesley 1993.

[10] Guy L. Steele, *"Common Lisp the Language, 2nd edition,"* Digital Press 1990.