HEURISTICS AS AN AID TO BACKTRACKING , A CLASSROOM PROJECT*

Dr. Gerald Wildenberg
Department of Mathematics, Computer Science and M/S/T
St. John Fisher College[1]
Rochester, NY 14618
716 385 8179
wildenbe@sjfc.edu

ABSTRACT

The knight's tour problem is discussed as a classroom example of a backtracking problem which requires heuristics to be effective.

Knight's tours by heuristically guided backtracking -- a classroom project.

The purpose of this paper is to show an excellent classroom example of backtracking combined with heuristics. We will see that straightforward backtracking may be an inadequate tool unless bolstered by heuristically guided choices. The example used can form the basis for a discussion of this topic in an Artificial Intelligence course or a course in Algorithms or a course in Problem Solving and even possibly in a first course in programming if time permits a discussion of backtracking techniques. Though my focus is not on the Object Oriented approach, this problem can certainly be treated from that point of view.

Perhaps a bit tangentially, let us first note that many problems from recreational mathematics lend themselves to computer solution. For example, it is not hard to write a

[1]I wish to thank the University of the West of England, Bristol, England, which generously provided me with the office space and computers I needed during the Spring of 2001 when the work on this paper was done.

program, using nothing more than for-loops and if-statements to do a brute force search for solutions to the old classic: SEND + MORE = MONEY, where each letter represents a different digit (nor is it terribly hard to solve by hand).

However when we get to problems with larger domains, more sophisticated methods, such as backtracking, may be needed. For instance, many texts give as a backtracking example the problem of putting eight queens onto a chessboard in such a fashion that no two occupy a common rank, file, or diagonal, that is to say that in chess parlance, no two "attack" each other. But what is often ignored is the fact that this turns out to be a computationally feasible method largely because the tree implicitly traversed is only eight levels deep.

The problem I will discuss is the old classic of generating a "Knight's Tour of the Chessboard". That is to say choosing a square on a chessboard, and then moving like a knight (see Figures I and II) to another, not yet visited square, until all the squares on the board have been covered. I will refer to this as the KTP (Knight's Tour Problem).

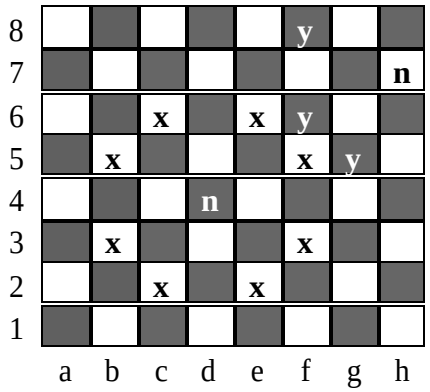

Figure 1 -- Knight (abbreviated as n in chess) on square d4 can move to any of the 8 squares marked x while the knight on h7 has only the three possibilities marked y.

56	**47**	**28**	**9**	**32**	**5**	**26**	**7**
29	**10**	**57**	**46**	**27**	**8**	**33**	**4**
60	**55**	**48**	**31**	**64**	**39**	**6**	**25**
11	**30**	**61**	**58**	**45**	**24**	**3**	**34**
54	**59**	**44**	**49**	**38**	**63**	**40**	**23**
15	**12**	**53**	**62**	**41**	**20**	**35**	**2**
52	**43**	**14**	**17**	**50**	**37**	**22**	**19**
13	**16**	**51**	**42**	**21**	**18**	**1**	**36**

Figure 2 -- An example of a Knight's Tour on an 8x8 chessboard.

Though a chessboard for the standard game of Chess is an 8x8 grid, we will also talk about chessboards of various sizes. (Caution: For boards of odd order, such as 5x5 or 7x7, solutions exist only for starting points whose coordinates total to an even number. I will later refer to starting squares which can possibly yield a solution to KTP as reasonable squares.) Finally I note that an instructor who wanted to introduce more novelty could easily apply the ideas discussed here to boards of various shapes, rectangular and nonrectangular and to pieces whose move was different than the traditional knight.

Before continuing, let's define backtracking as the process of finding a path in a tree which is a solution to a problem by going as far down the tree as possible but, if a solution is not found, returning to the parent node and then trying the next child. It is the return to the parent node which gives the method its name. More succinctly, backtracking is a depth first search of a (usually not explicitly stored) tree.

Our first thought might be that backtracking is a natural for solving the KTP since when we try to do a knight's tour, say by hand, we will frequently get to positions where we have no further moves but have not yet occupied all the squares of the board. At that point we can backtrack and try an alternative move not yet tried. Most of the paths down the tree of sequences of knight's moves will end when the knight has no further moves. In looking for a knight's tour, we are looking for a path of maximal length -- 64 for an ordinary 8X8 chessboard. Thus we are searching a tree with 64 levels. Since there are up to eight possible branches at each position, we can see that the tree being searched is very large indeed.

For the knight's tour problem (and many similar problems), the pseudocode for a recursive implementation of backtracking is essentially:

```
main(){
initialize();
knightstour(1);
printboard();
}
function knightstour(n){
  if (n == NUM_SQUARES){
      print_board;
      return;
  }
  generate a list of legal moves;
  // This loop provides the backtracking.
  for each possible move do{
    mark board with move;
    knightstour(n+1);
            remove mark of move;
           }
  return;
```

}

This brings us to the question as how we generate the list of moves. The two most primitive methods are to: 1. Decide on some order for trying the possible moves (such as clockwise from 1:00) and, at each node, try the various possibilities. 2. Make a list of the possible moves in random order.

In order to help students appreciate both the power and deficiencies of a straight backtracking algorithm in solving KTP, I suggest that students start with strategy 1. or 2. and try it on various sizes of chessboards. When students do this, they find that solutions are quickly and easily found for 5x5 boards but that the method is rather unpredictable for larger boards, sometimes producing a quick solution, but more often taking many millions of backtracking calls before finding a solution. For larger boards, random backtracking turns out to be computationally impractical.

If one tries to find solutions to the KTP by hand (a recreation which I enjoyed during more than a few hours of tedious classes in my youth), one quickly finds that moving the knight randomly is an ineffective method. This suggests that some heuristic (which can be informally defined as a "rule of thumb") method of ordering the moves might lead to a faster solution. There are many reasonable heuristics to try. Here are a few possibilities (continuing the numbering from above): 3. Go to an edge if that is possible. 4. Go to a corner if possible. 5. Go as close to an edge as possible. 6. Go as close to a corner as possible. Needless to say, "close" needs to be defined with care in heuristics 5. and 6. For 6. one possibility (perhaps the most likely) is to use the sum of the distances to the two nearest edges. The multitude of possible heuristics offers the student the opportunity to try to devise a rule which will produce solutions quickly in as large a board as possible. Heuristics 3.-6. only scratch the surface -- there are many other possibilities.

When combined with backtracking, these heuristics can be used to order the list of possible moves. The hope is that by trying the most promising moves first (according to some heuristic) we will shorten the time involved in searching the tree of possibilities until a solution is found. Some of the heuristics lead to an ordering of the list of possible moves; others simply give a move to try first and thus leave the remainder of the possibilities to be tried randomly.

How will we evaluate the effectiveness of such heuristics? Because of the wide variability in the number of calls generated when we choose various starting squares, I suggest to students that they count the number of backtracks required and try using each of the possible starting squares (possibly reducing this by utilizing the eightfold symmetry of the square) and averaging the results. This produces a reasonable evaluation of the effectiveness of various heuristics.

Some readers will have been wondering when I would finally discuss the famous heuristic known as Warnsdorff's Rule (hereafter WR). My first acquaintance with this came many years ago when I saw it mentioned briefly in the problems section of Horowitz and Sahni's famous textbook on Data Structures. The heuristic dates back to 1823. WR states: Move to the square from which there are the fewest moves. Though this heuristic is very powerful, my personal experience is that it is not at all obvious until pointed out. That is fortunate as part of the appeal of this problem as a classroom project is that students have an opportunity to use

their own ingenuity and imagination in devising a heuristic. However after learning WR, it is easily seen to be a stronger version of several of the heuristics suggested above. It is important to note however that while WR will frequently lead to a solution without using any backtracking, in an unmodified form, it is not guaranteed to give a solution.

In order to test some of the heuristics, I used the following method. I decided (somewhat arbitrarily, but based on what I regarded as a "reasonable" time) to call any run which required more than 10,000,000 calls to the backtracking routine a failure. I tested each method from all the reasonable squares of a chessboard and recorded how many were successful and what the average number of calls were in the successful outcomes. Note that your implementation may well get somewhat different results due to effect described in footnote [2].

The power of using a sensible heuristic can be easily seen in the 6x6, 7x7 and 8x8 case where the random heuristic is largely a failure while the "distance to corner" heuristic produces lots more successes. And the value of very good heuristics can be seen in the 8x8, 9x9 and 10x10 case where WR produces many more successes than other heuristics.

I would welcome feedback from others who try this project in the classroom or who perform further experiments with various heuristics.

Figure 3 -- Table of results of several heuristics combined with backtracking.

	Heuristics		
	Randomly ordered moves.	**Moves ordered according to "distance"[3] from corner, closest tried first.**	**Moves ordered a la Warnsdorff, squares with fewest exits examined first.**

[2]Even when the heuristic provides an ordering, I have observed that there is an element of accident due to the way ties are broken. That is to say if two moves get the same rating by the heuristic, they must still be tried in some arbitrary order. This may have considerable effect. Of course a modification of such a heuristic could provide a tie-breaking mechanism.

[3]The distance was calculated as the sum of the distances to the closest horizontal and closest vertical edge, that is by what is sometimes called the taxi-cab metric.

5x5	0 failures	0 failures	0 failures
	Average: 139,112	Average: 45 (Many were done in the minimum of 25.)	Average:339 (Many were done in the minimum of 25.)
6x6	25 failures	0 failures	0 failures
	Average: 2.4 million	Average: 672	Average: 37 (Many were done in the minimum of 36.)
7x7	23 failures (out of 25)	0 failures	0 failures
	Average: 3.37 million	Average:52,387	Average: 152,840
8x8	64 failures	16 failures	2 failures
	No average	Average: 451,371	Average: 73
9x9	Not tried	22 failures (out of 41)	0 failures
		Average: 3.10 million	Average: 496
10x10	Not tried	Not tried	1 failure
			Average: 101

BIBLIOGRAPHY

1. The following URL provides a discussion of Warnsdorff's Rule, and a link to item 2.: http://w1.859.telia.com/~u85905224/knight/eWarnsd.htm

2. This URL contains a brief paper giving an improvement to Warnsdorff's Rule: http://sunny.mpimf-heidelberg.mpg.de/people/roth/Mma/Knight_1_0_0.html

3. Fundamental's of Data Structures in Turbo Pascal by Horowitz and Sahni. pp 177-180. Though not the edition referred to in this article (that one has disappeared), this contains a brief discussion of the programming of Warnsdorff's Rule.