USING BACKTRACKING TO SOLVE THE SCRAMBLE SQUARES® PUZZLE

Keith Brandt, Kevin R. Burger, Jason Downing, Stuart Kilzer
Mathematics, Computer Science, and Physics
Rockhurst University,
1100 Rockhurst Road, Kansas City, MO 64110-2561
<keith.brandt@rockhurst.edu>, <kevin.burger@rockhurst.edu>

ABSTRACT

Constructing computer programs that solve problems (and puzzles) is central to the field of Artificial Intelligence (AI). In this paper we present an interesting puzzle and show how it can be solved using depth first search with backtracking. We feel this problem is well suited for undergraduate student projects, as it clearly demonstrates the power of backtracking when working with very large search trees. We conclude with several extensions to the original puzzle that would provide excellent research opportunities for undergraduate students.

This project was supported in part by a Rockhurst University School of Arts and Sciences Dean's Summer Undergraduate Research Fellowship (Summer, 2001).

INTRODUCTION

Constructing computer programs that solve problems (and puzzles) is central to the field of Artificial Intelligence (AI). Work has been done in AI to find efficient and optimal *search algorithms* for solving various problems. In this paper we present an interesting puzzle and show how it can be solved using depth first search with backtracking.

Scramble Squares® is a puzzle created and marketed by b.dazzle, inc., [1]. The puzzle consists of nine squares, each of which contains four halves of various images. Figure 1 shows the nine unarranged squares of one of their puzzles. The object of the puzzle is to arrange the squares in a 3×3 grid such that the "scrambled" images are aligned, thus solving the puzzle. Similar puzzles are marketed by MindWare (Squzzle™) [2] and the DaMert Company (3D Squares™) [3].

The puzzle is very complex. Indeed, there are $4^9 \cdot 9! = 95{,}126{,}814{,}720$ different arrangements of the nine pieces. This puzzle is actually an instance of what is known in AI as a *constraint satisfaction problem* (CSP). These problems have been studied extensively and continue to be an important area of research. Many of them are known to belong to the complexity class NP-complete and thus are generally very

difficult to solve for large values of *n*. Nonetheless, there exist algorithms which are better than others for attacking CSP's. Two such algorithms include *depth first search with forward checking* and *depth first search with backtracking*.

In this paper, we describe a depth first search with backtracking algorithm used to solve the puzzle, discuss the performance of the algorithm, and consider questions for further study. We feel this problem is well suited for student projects, as it clearly demonstrates the power of backtracking when working with a very large search tree. For descriptions and examples of other applications of search algorithms, see [4] and [5].

Figure 1. Unarranged Squares of the Scramble Squares Puzzle

THE SETUP

Squares are placed into the 3×3 grid, one at a time, in the order shown in Figure 2. Given a partial solution consisting of *k* squares, the remaining squares are systematically tested to see if any fit in the (*k* + 1)-st position. If so, the process continues. If not, the *k*-th square is removed (backtrack), and the other remaining squares are tested in the *k*-th position.

7	8	9
6	1	2
5	4	3

Figure 2. Initial Placement of the Squares in the 3×3 Grid

The square in position 1 is not rotated, so redundant solutions that correspond to rotating the entire grid are eliminated. At each of the remaining steps, the squares being considered are tested in four different orientations (rotations). Thus, with this restriction, there are $4^8 \cdot 9! = 23{,}781{,}703{,}680$ possible arrangements of the puzzle squares.

The structure of the search tree is shown in Table 1. Since the first square is not rotated, there are 9 nodes at the first level. In general the number of nodes at the k-th level of the tree is given by $9 \cdot (8 \cdot 4) \cdot (7 \cdot 4) \cdots ((10 - k) \cdot 4) = 9! \cdot 4^{k-1} / (9 - k)!$. There are a total of 30,563,311,978 nodes in this search tree.

Table 1. Number of Nodes at Each Level of the Search Tree

Level	Number of nodes
Root	1
1	9
2	288
3	8,064
4	193,536
5	3,890,720
6	61,931,520
7	743,178,240
8	5,945,425,920
9	23,781,703,680

THE ALGORITHM

The squares are numbered 1 through 9 and are stored in an array PIECE. Orientations are numbered 0 though 3. The algorithm uses a 1 by 9 array USED of zeros and ones that keeps track of the pieces being used in the current (partial) solution. The partial solution is stored in a 2 by 9 array SOLUTION that contains the square numbers in the correct order and the orientation of each square. For example,

$$\text{SOLUTION} = \begin{matrix} 7 & 5 & 1 & 9 & 4 & 3 & 8 & 2 & 6 \\ 0 & 3 & 1 & 0 & 0 & 3 & 1 & 4 & 2 \end{matrix}$$

means put PIECE(7) with orientation 0 in the first location, put PIECE(5) with orientation 3 in the second position, and so on.

We use recursion to look for solutions. When we have a partial solution consisting of k squares, the procedure SOLVEPUZZLE looks for a square that will fit in the $(k + 1)$-st position. Figure 3 shows the basic structure of SOLVEPUZZLE.

```
Procedure SOLVEPUZZLE (k : integer)
  If k < 10 Then
    While there are more pieces to consider
      Select a new piece PIECE(n) for consideration
      Set ORIENTATION := 0
      While ORIENTATION < 4
        If new piece, in cur ORIENTATION, fits in position k Then
          SOLUTION(1, k) := n
          SOLUTION(2, k) := ORIENTATION
          USED(n) := 1
          SOLVEPUZZLE(k + 1)
          USED(n) := 0
        Else
          Increment ORIENTATION
        End If
      End While
    End While
  Else (when k = 10, we have found a solution)
    Display SOLUTION
  End If
End Procedure
```

Figure 3. The Recursive Backtracking Procedure for Solving the Puzzle

RESULTS

By placing counters within SOLVEPUZZLE, we can keep track of the number of times this procedure is called at each level of the search tree. These numbers are shown in Table 2. SOLVEPUZZLE was called three times at level 10, so there are three distinct solutions to this Scramble Squares puzzle. The table also reveals just how effective backtracking can be: SOLVEPUZZLE was called a total of just 672 times to search a tree with over 30 billion nodes.

Table 2. Number of Recursive Calls to SOLVEPUZZLE at Level *k*

k	Calls at level *k*
1	1
2	9
3	37
4	140
5	80
6	203
7	62
8	116
9	21
10	3

THE JAVA APPLET

We have written a Java applet that implements the puzzle [6]. The applet is initialized with a puzzle configuration corresponding to Figure 1. The user can attempt to solve the puzzle manually or can let the computer solve the puzzle using the

recursive backtracking algorithm. The applet will also create random 3×3, 4×4, and 5×5 puzzles, which can be quite challenging to solve manually.

QUESTIONS FOR FURTHER STUDY

The original puzzle is very interesting, but augmenting the puzzle and varying the parameters will lead to many other interesting questions. Here are some problems worth considering:

1. Are there more efficient ways to solve the Scramble Squares puzzle? References [2] and [3] describe several other search techniques that might be used to solve the puzzle.
2. By rearranging the images on the squares, is it possible to construct a puzzle that has a unique solution?
3. Construct and solve a similar puzzle for a 4×4 grid. This puzzle would allow over 10^{22} different arrangements of the squares. How long would it take to solve a 5×5 puzzle?
4. Construct and solve a two-sided puzzle (whose pieces can be flipped over as well as rotated).
5. Construct and solve a similar puzzle consisting of six squares that must be arranged on the faces of a cube. Are any six of the existing nine Scramble Squares pieces a solution to such a puzzle?
6. Construct and solve similar puzzles where equilateral triangles are arranged on the faces of an octahedron (or, for the ambitious, an icosahedron) or where pentagons are arranged on the faces of a dodecahedron.

ACKNOWLEDGEMENTS

We would like to express our appreciation to the Rockhurst University Arts and Sciences Dean's Office for its support of this undergraduate research project.

REFERENCES

1. *Scramble Squares*®, 2001. [online]. b-dazzle, inc., [cited 6 Nov 2001]. <www.b-dazzle.com>. Scramble Squares is a registered trademark of b-dazzle, inc.
2. *Squzzle*™, 2000. [online]. MindWare, [cited 7 Jan 2002]. <www.mindwareonline.com>. Squzzle is a trademark of MindWare.
3. *3D Squares*™, 2001. [online]. DaMert Company, [cited 7 Jan 2002]. <www.damert.com/pages/3dsquares.html>. 3D Squares is a trademark of the DaMert Company.
4. Michalewiz, Z., Fogel, D. B., 2000. *How to Solve It: Modern Heuristics*, Springer-Verlag, Berlin, Germany.
5. Russell, Stuart J., Norvig, Peter, 1995. *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
6. Burger, Kevin R., 2001. [online]. Rockhurst University, [cited 7 Jan 2002]. <cte.rockhurst.edu/burgerk/research/scramble.html>.