Teaching Proofs and Algorithms in Discrete Mathematics with Online Visual Logic Puzzles

JOHN CIGAS
Rockhurst University

WEN-JUNG HSIN
Park University

Visual logic puzzles provide a fertile environment for teaching multiple topics in discrete mathematics. Many puzzles can be solved by the repeated application of a small, finite set of strategies. Explicitly reasoning from a strategy to a new puzzle state illustrates theorems, proofs, and logic principles. These provide valuable, concrete examples, in addition to the algebraic proofs that make up most of the examples in textbooks. Creating specialized, pedagogical applets allows more automation of repetitive tasks as well as a framework for discussing and implementing algorithms for solving the puzzles. One such applet is presented here and several possible exercises using it are described.

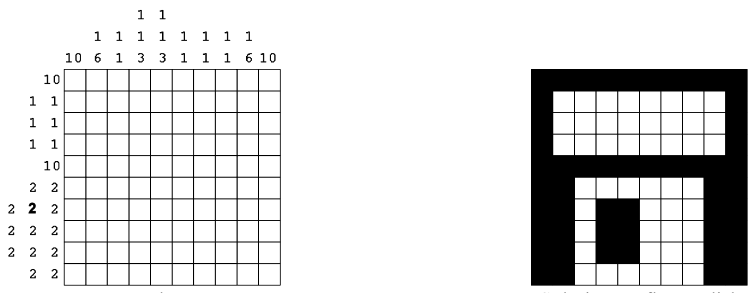
Categories and Subject Descriptors: G.2.0 [**Discrete Mathematics**]: General; K.3.2 [**Computers and Education**]: Computer and Information Science Education - *Computer science education*; D.2.3 [**Software Engineering**]: Coding Tools and Techniques – *Object-oriented programming*
General Terms: Algorithms
Additional Key Words and Phrases: Logic, education, proof

1. INTRODUCTION

Many students of introductory discrete mathematics seem to struggle with mathematical proofs. There are many reasons for this, including weak algebraic skills. Since many textbook proofs are algebraic in nature, these students get lost in the logical argument simply because they do not see the algebra. Good pedagogy dictates that nonalgebraic proofs should be introduced as well. One example seen in many texts uses induction to prove that a $2^n \times 2^n$ chessboard with any one square removed can be covered by L-shaped tiles that cover three squares at a time [Rosen 2003; Dossey et al. 2002]. Section 2.1 shows this in more detail. This is a nice example of a visual, non-algebraic proof. However, not everyone wants to introduce proofs using induction.

Another technique is to use a visual puzzle, such as the computer game Minesweeper [Lock and Struthers 1999; Greenwald 2003]. Students are given a partially completed Minesweeper board and then asked to prove whether a certain cell is definitely a bomb or definitely empty. Section 2.2 shows this in more detail. Aside from the premise of the game, it is an excellent tool for describing proofs, since students are either already familiar with the game or can learn the game with a little practice and experimentation. The logical proof is then just a matter of formalizing the reasoning they have already developed.

Authors' addresses: J. Cigas, 1100 Rockhurst Rd., Kansas City, MO 64110; email: cigas@acm.org; W. -J. Hsin, 8700 NW River Park Dr., Parkville, MO 64152; email: wen.hsin@park.edu.



Puzzle Solution - a floppy disk

Figure 1

This paper takes the concept of using non-algebraic proofs and extends it by using visual logic puzzles to introduce logic, proofs, and algorithms. Visual logic puzzles benefit students by strengthening their reasoning and developing their concentration, while at the same time giving them an enjoyable experience (usually) and a visually pleasing result. The goal of this type of puzzle is to fill in squares in a grid to create a picture. The numbers surrounding the grid are the specifications of the size of the black blocks that are in each row and column of the grid. One such puzzle is shown in Figure 1.

The remainder of this paper is organized as follows. Section 2 outlines some examples of visually oriented, non-algebraic proofs. Section 3 describes other attempts to introduce logic and proofs and then shows the advantages of using these particular visual logic puzzles to introduce proofs in a discrete mathematics course. Section 4 shows how to use these puzzles to introduce algorithms, both the concepts and the notation. Section 5 presents a prototype applet that automates some of the drudgery of writing down the steps in the proof and provides a framework for students to develop, code, and test their own strategy algorithms for solving a puzzle. Section 6 describes some of the activities possible using these puzzles in introductory mathematics and programming classes, and finally, Section 7 lists some resources available for incorporating these puzzles in the classroom.

2. EXAMPLES OF NON-ALGEBRAIC PROOFS

Here we briefly describe some of the non-algebraic problems described in the introduction. More details on the actual proofs are found in the original sources and later in this paper.

2.1 Tiling a chess board

This problem is to show that a $2^n \times 2^n$ square board can always be covered with L-shaped tiles, except for one square. This must work regardless of which square is chosen to be left uncovered. The proof uses induction. For the base case of n=1, enumerating the 4 possible orientations of the L-shaped piece is all that is necessary (Figure 2).

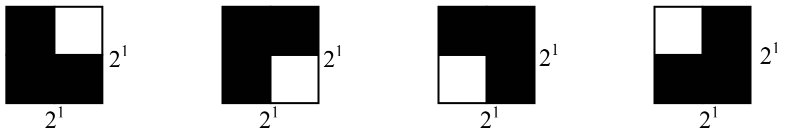


Figure 2

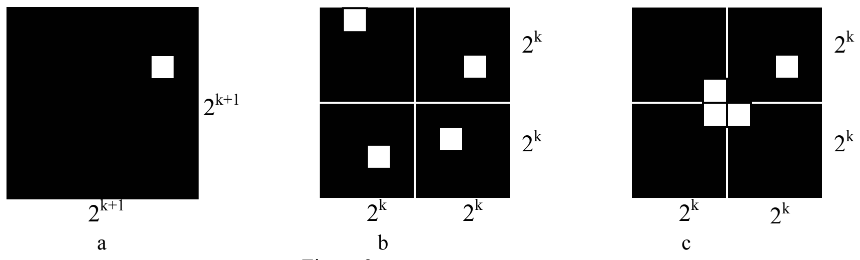


Figure 3

Figure 3a shows the inductive case goal, a $2^{k+1} \times 2^{k+1}$ board with a single square removed. We note that this board can be partitioned into four $2^k \times 2^k$ boards. From the inductive hypothesis, each of these $2^k \times 2^k$ boards can be tiled, except for one square in each (Figure 3b). But, the tiling works no matter which square is removed, so we can arbitrarily choose the uncovered squares to be in the center of the board (Figure 3c). From here, it is obvious that the entire 2^{k+1} x 2^{k+1} board can completely covered, except for a single square.

2.2 Minesweeper

A proof using the game Minesweeper starts with a partially exposed board, such as the one in Figure 4a. The number in a cell represents the number of mines that are hidden in the cells immediately surrounding that cell. Students are asked to reason about whether the cell marked with a double circle is a mine or not. The two-step proof starts with the cell marked by a single circle. The 1 in this cell means that exactly one adjacent cell contains a mine, and since there is only one uncovered cell (to the upper right), that uncovered cell must be a mine. Figure 4b shows this cell marked as a mine. Now, all the cells marked with '?'s cannot contain mines, since they are all adjacent to a cell with a 1 that is also adjacent to the marked cell. Therefore, we can definitively state that the double-circled cell is not a mine.

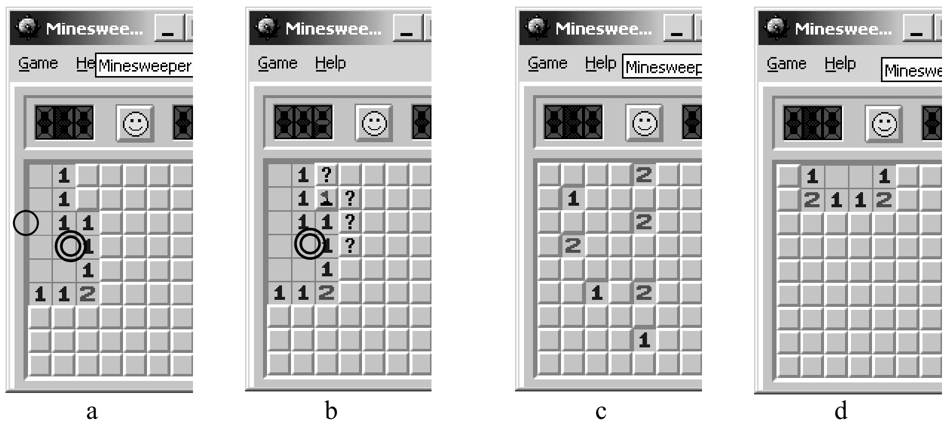


Figure 4

3. LOGIC AND PROOFS

3.1 Proofs with textual logic puzzles

Logic in most discrete mathematics textbooks is fairly dry, although one text due out in Spring 2005 [Ensley and Crawley 2005] emphasizes puzzles and games as the context for many examples. Over the years we've experimented with using textual logic puzzles to try to motivate students. These have a form like "There are four people, each with a different pet, and each with a different color hat. Bill doesn't have a red hat or a goldfish. The woman with the dog has a blue hat. …" The goal is to find which person has which pet and which color hat by reasoning from the clues. Although these are good puzzles to solve, we found them less than ideal in the classroom. First, explaining them took a bit of effort and class time, especially in regard to solution strategies. Most students have never seen them before and found the format difficult to grasp quickly. Second, students tended to get confused in their solutions, and would often not even realize when they made mistakes. Third, even with a correct solution, it wasn't easy for any student to write down the reasoning behind the solution, because there were no clear, consistent strategies to apply.

3.2 Proofs with on-line games

Games like Minesweeper, on the other hand, provide a much more familiar context for discussing proofs and strategies. However, there are several downsides to using this particular game. First, the nature of the game, locating hidden mines and not getting blown up, is not appealing to everyone. Second, the original Minesweeper game randomly assigns mines and it cannot always be solved without some guessing. Figure 4c illustrates that even after nine random selections, there is still nothing we can deduce about the position of mines on the board. Other positions, such as in Figure 4d, may arise after several steps into the game. While this enhances the game playing experience, it is frustrating to spend time constructing a proof, only to be stymied by a position from which further progress is impossible without a guess. The end result is that it is necessary to use pictures of a single game state and then have students reason from that picture, as already described. This is a little awkward since it is only one step in the process. There is no way of continuing with the proof, since the rest of the states are unknown. While it is possible to find a clone of the minesweeper program that allows total control of mine placement, we opted to pursue using visual logic puzzles instead.

3.3 Proofs with visual logic puzzles

There are several types of visual logic puzzles, but the one we concentrate on in this paper goes by the names such as Nonograms™, Griddlers™, Paint by numbers, Pic-a-pix, and many more. The goal of this type of puzzle is to fill in squares in a grid to create a picture. Figure 5 shows a very simple logic puzzle.

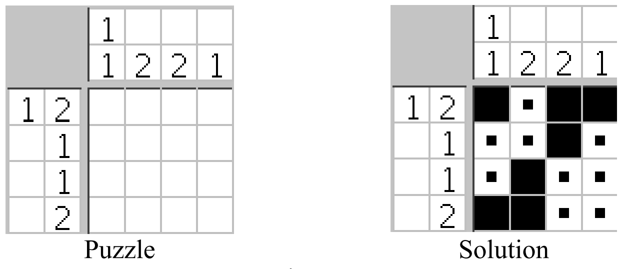
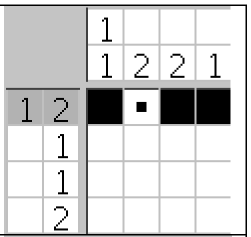
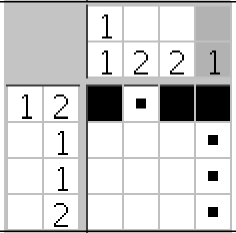
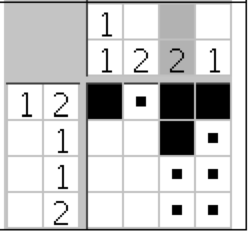


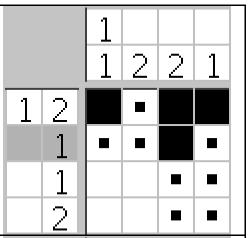
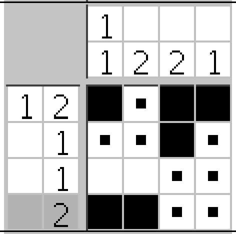
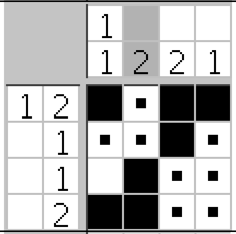
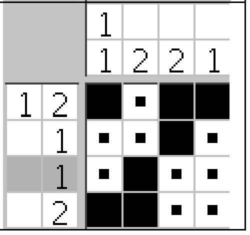
Figure 5

The grid on the left is the puzzle and the grid on the right is the solution. The numbers surrounding the grid are the *specifications*. They describe the number and size of black blocks for each row and column. We will refer to them using <>. For example, the <2> in the second column means that there is a single block of two black squares somewhere in that column. The <1 2> in the first row means there is one block of a single black square, followed by one or more empty spaces, followed by a block of 2 black squares. Blocks of black squares are always separated by at least one empty space.

Puzzle solvers apply a number of logical *strategies* to either mark squares as filled (black) or empty (dot – a legacy from paper solutions). Unknown squares are blank. For example, if a specification says there must be exactly <1> black square on a row (like row 2 in Figure 5), and if there is already 1 black block on that row, then all the rest of the squares must be dots. Another example, one often used to get a puzzle started, is that if a specification says there must be <n> black squares in a block and a row contains less than 2n squares, then some of the center squares must be filled in black. To illustrate this overlap strategy, suppose that a row has 4 squares and the specification says there is a single block of 3 black squares. This means that the two center squares must be black, while the squares at each end of the row remain unknown.

The steps in solving the puzzle in Figure 5 are documented below. This is the answer key to an assignment where we ask students to show the steps and reasoning required to create the picture on the right in Figure 5. We use <> to indicate a specification and (n,m) to indicate a square at (row n, column m) in the grid.

1. Row 1: The specification <1 2> row requires at least 4 squares (1 black, 1 empty, 2 black). Since there are only 4 squares in the row, this must be the complete pattern for this row. Therefore, square (1,1) is black, (1,2) is empty, and (1,3) and (1,4) are both black.	
2. Column 4: The specification <1> requires exactly 1 black square, which is already filled at location (1,4), therefore the rest of this column must be empty.	
3. Column 3: The specification <2> requires exactly 2 consecutive black squares. Since square (1,3) is black and at the edge of the grid, then square (2,3) must also be black and the remaining squares in the column must be empty.	

4. Row 2: The specification <1> requires exactly 1 black square, which is already at location (2,3), therefore, the rest of this row must be empty.	
5. Row 4: The specification <2> requires exactly 2 consecutive black squares. Since square (4,3) and (4,4) are empty (from steps 2 and 3), the only remaining possibility is for squares (4,1) and (4,2) to be black.	
6. Column 2: The specification <2> requires 2 consecutive black squares. Since square (4,2) is black and at an edge, therefore square (3,2) must be black.	
7. Row 3: The specification <1> requires exactly 1 black square, which is already at location (3,2), therefore, the rest of this row must be empty.	

At this point all squares are either definitively black or empty, so the puzzle is solved and the proof that these specifications generate this picture is complete.

This exercise also shows a valid argument. If one wanted more formality, then one could assign names to the specifications for each row and column (r1, r2, …, c1, c2, …) , and names to each square ($a_{1,1}$ means square (1,1) is black, $\neg a_{1,2}$ means square (1,2) is empty). Rewriting the proof using this notation along with logical operators (and, not, implication) would yield a valid argument notationally similar to what appears in most texts.

$r1 \rightarrow a_{1,1} \wedge \neg a_{1,2} \wedge a_{1,3} \wedge a_{1,4}$

$c4 \wedge a_{1,4} \rightarrow \neg a_{2,4} \wedge \neg a_{3,4} \wedge \neg a_{4,4}$

$c3 \wedge a_{1,3} \rightarrow a_{2,3} \wedge \neg a_{3,3} \wedge \neg a_{4,3}$

…

We haven't done this in class because of the mechanical drudgery required to translate and copy the notation. However, this is the type of notation that could be easily automated in a program, as will be described later. Seeing one concrete example like this strongly reinforces the concept of a valid argument for students.

3.4 Limitations

Our approach is to provide a context for introducing proofs that is simple and straightforward. Toward that end, we assume every puzzle has a unique solution that can be obtained by logical reasoning. This ignores the fact that some puzzles are unsolvable and others require guessing and backtracking in order to find the unique solution. One suggested activity in Section 6 is to have students create a puzzle that stumps a set of strategies, thus prompting a discussion of these other issues. However, this is well beyond the scope of introducing proofs.

4. ALGORITHMS

4.1 Learning algorithms in discrete mathematics

Algorithms are often introduced in discrete mathematics courses as part of the study of asymptotic complexity, i.e. O(), Ω(), and Θ(). Especially for students without a programming background, this requires learning new notation, new concepts of decisions, loops and possibly parameters, and then learning to analyze these algorithms. Implementing individual strategies to solve visual logic puzzles provides a way of introducing algorithms in which the problem space is visual and familiar.

Even though these particular puzzles use a two-dimensional grid, most strategies operate on a single row or column at a time. For algorithm development, this means linear data structures like lists, and simple loops to process the elements in the lists.

4.2 Left edge strategy as a beginning algorithm

Consider the strategy of filling in a block of black squares if the left edge of a row is already filled in. In words, the *Left Edge* strategy can be expressed as: If the leftmost square in a line is black, then this must be the beginning of a block of black squares. The first number in the specification list tells how many squares are in the block, and the following square, if it exists, must be empty, since there is always at least one empty space between blocks. This is illustrated in Figure 6 below. We label the squares in the line L_1 .. L_n, and the specifications S_1 .. S_m.

Using notation adapted from Rosen, the corresponding algorithm is shown below.

```
procedure leftedge ($L_1, L_2, \ldots L_n$: squares in a line {black, empty, unknown};
                    $S_1, S_2, \ldots S_m$: integer specifications)
    if ($L_1$ = black) then
    begin
        $i := 2$
        while $i \le S_1$
        begin
            $L_i$ := black
          $i := i + 1$
        end
        if $i \le n$ then $L_i$ := empty
    end
```

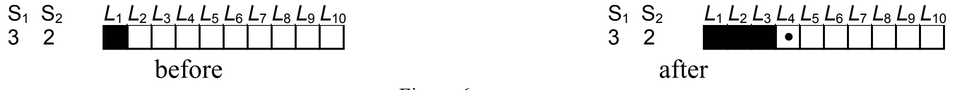

S1 S2
3 2
L1 L2 L3 L4 L5 L6 L7 L8 L9 L10
before
S1 S2
3 2
L1 L2 L3 L4 L5 L6 L7 L8 L9 L10
•
after

Figure 6

While lengthier than other introductory algorithms, like finding the maximum value in a list of integers or searching for a value in a list of integers, this algorithm corresponds almost exactly to the process used by a human in filling in squares in the puzzle. Because of this correspondence, it puts the components of decision and looping in a familiar and easier to understand context.

4.3 Right edge strategy

While superficially similar to the left edge strategy, this introduces many new details. In words, the strategy is: If the rightmost square in a line is black, then this must be the end of a block of black squares. The last number in the specification list tells how many squares are in the block, and the preceding square, if it exists, must be empty, since there is always at least one empty space between blocks. Turning this into an algorithm requires them to find the leftmost cell in the block at index $(n - S_m + 1)$ and then formulate a loop that either starts or ends at that index.

Alternatively, students could see that with some helper functions, this can be expressed as

```
reverse the list of specifications
reverse the list of line cells
apply the left edge strategy
reverse the list of line cells
reverse the list of specifications
```

4.4 Other strategies

There are several other possible strategies for introductory algorithms:

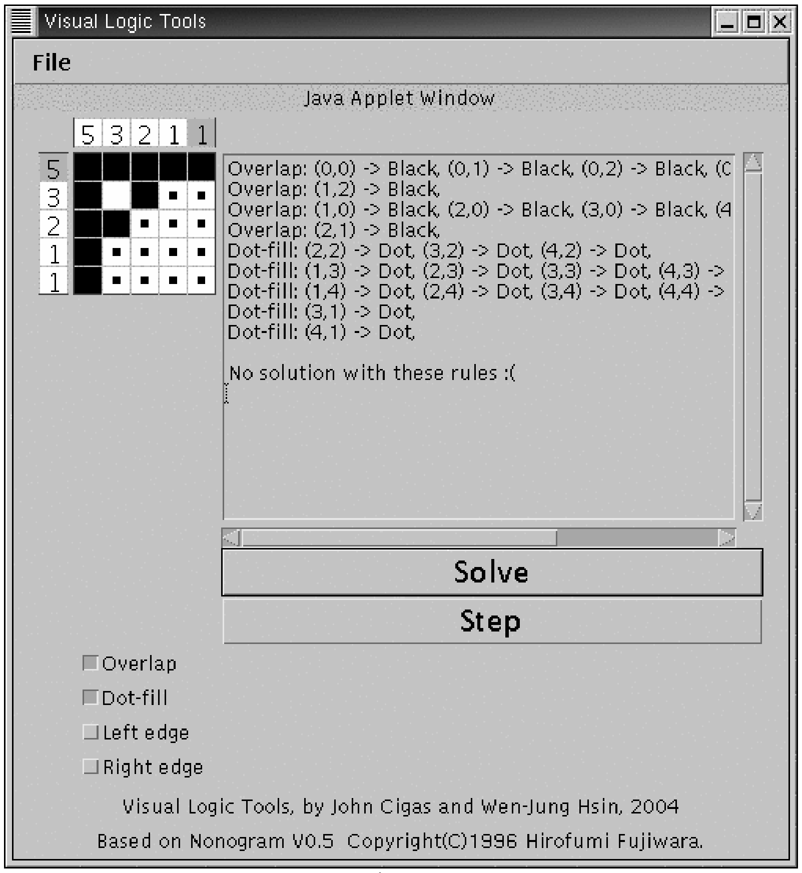
- dot-fill: If the total number of black squares matches the total of the sizes in the specification, then the remaining squares must be empty.
- overlap: If the number of black squares in the specification is more than half the length of the line, then some squares must always be filled in. This strategy is easier to construct if one starts off assuming there is only one block of black squares in a line. For example, if a line contains 10 squares and the specification says there is a single block of 9 black squares, then we know for sure that all squares except the first and last must be filled in.
- all-fill: If the sum of all the *m* specifications and the (m–1) required spaces between the blocks equals the length of the line, *n*, then the specifications exactly describe the line.

4.5 Limitations

These strategies are usually sufficient for an introduction to algorithms in a discrete mathematics class. Once students become familiar with the notion of an algorithm and the notation, then traditional algorithm examples, which are better suited to simple asymptotic analysis, can be introduced. In general, these simplistic puzzle strategies are necessary, but not sufficient, to solve most of these types of visual logic puzzles.

5. PEDAGOGICAL APPLET

Logic and proofs can be taught using printed puzzles or interactive ones on the web. The benefit to the online ones is that there is often an undo and a check button, so students can be more successful more quickly. However, most on-line games are meant as


Visual Logic Tools
File
Java Applet Window
5 3 2 1 1
5 3 2 1 1
Solve
Step
Overlap
Dot-fill
Left edge
Right edge
Visual Logic Tools, by John Cigas and Wen-Jung Hsin, 2004
Based on Nonogram V0.5 Copyright(C)1996 Hirofumi Fujiwara.

```
Overlap: (0,0) -> Black, (0,1) -> Black, (0,2) -> Black, (C
Overlap: (1,2) -> Black,
Overlap: (1,0) -> Black, (2,0) -> Black, (3,0) -> Black, (4
Overlap: (2,1) -> Black,
Dot-fill: (2,2) -> Dot, (3,2) -> Dot, (4,2) -> Dot,
Dot-fill: (1,3) -> Dot, (2,3) -> Dot, (3,3) -> Dot, (4,3) ->
Dot-fill: (1,4) -> Dot, (2,4) -> Dot, (3,4) -> Dot, (4,4) ->
Dot-fill: (3,1) -> Dot,
Dot-fill: (4,1) -> Dot,

No solution with these rules :(
```

Figure 7

entertainment and not as teaching tools. To enhance the pedagogical value of these visual logic puzzles, we have developed a prototype of an applet for use in the classroom. This prototype is based on a nonogram game-playing applet written by Hirofumi Fujiwara [Fujiwara 1996]. A screenshot of our applet is shown below in Figure 7.

For interactive use, our applet allows a student to select a puzzle from the `File` menu, use `check boxes` to select which strategies to apply to the puzzle and then use the `Step` button to step through each row and column applying the selected strategies. All transformations are logged to the large text area. There is also an option for solving as much of the puzzle as possible by clicking the `Solve` button. Figure 7 shows that this puzzle can almost, but not quite be solved by using just the first two strategies, overlap and dot-fill, discussed earlier in the paper.

In addition to the interactive component, the applet provides a ready-made framework for students to test their own implementations of a strategy, from a simple additional special case to a sophisticated, integrated algorithm. Adding a new strategy to the framework entails writing a class with a constructor and an apply() method, adding one line to the main applet (this can be automated later), and recompiling. Code for the left edge strategy is shown below.

```
class LeftEdgeStrategy extends NonogramStrategy {

public LeftEdgeStrategy() {
     super ("Left edge",
            "Fills in any line that has a square touching a left
         edge.");
}

public int[] apply(int [] line, int [] spec) {
     int j;
     int []res = dup(line);

     if (res[0] == FILLED) {
        for (j=1; j<spec[0]; j++)  // fill in the black squares
             res[j] = FILLED;
        if (j < res.length)    // add the dot after the black squares
             res[j] = EMPTY;
     }
     return res;
   }
}
```

For teaching algorithms, differentiating left and right as separate cases works very well. For coding, it is a bit redundant. As students solve these problems, they can start to see some symmetry. Anything that applies to the left also applies to the right, but with different bounds and direction. This might mean parameterizing a method, or possibly just reversing lines and specifications, solving, and then restoring back to the original order. While coding would be minimal in a discrete mathematics course, this framework does lend itself well to assignments in some of the introductory programming classes.

6. STUDENT ACTIVITIES

Here briefly, are a few of the activities and exercises that are possible using these visual logic puzzles as subject matter:

Without any specialized tools	
Activity	Benefit
Solve a visual logic puzzle, either on-line or on paper	• Introduction to logic, especially implication • Plain old fun
Write down steps in a solution	• Logical arguments and proofs
Develop strategies, like left-edge, dot-fill and overlap	• Algorithm development

Using a specialized applet[1]:	
Activity	Benefit
Automatically solve – view log of applied strategies	• See a non-algebraic proof
Convert log to formal notation	• See the proof in the same format as in text books

[1] Not all activities/exercises are currently supported by the prototype applet

Solve nonogram by hand and generate a log of strategies that apply to each change	• See a proof of a student solution
Given an algorithm that has been coded, students step through the applet to find flaws/limits of the algorithm	• Testing • Tracing
Given a set of strategies, try to find a nonogram that stumps the strategies	• Thoughtful analysis of algorithms • Testing
Develop, code, and step through new strategies	• Practice designing, coding, and testing in an OO framework

7. SOURCES

There are many sources for logic puzzles available on the web for students to use. Several sites with numerous puzzles listed below.

http://www.conceptispuzzles.com	Print and online versions of: • pic-a-pix: as described in this paper • fill-a-pix: minesweeper like clues • link-a-pix: connect up pairs of numbers
http://www.griddlers.net	User contributed puzzles • griddlers: as described in this paper • triddlers: cells are triangles, there are 6 axes for clues Workshop for creating your own puzzles
http://www.unixpapa.com/pbn/	User contributed puzzles Create your own puzzles
http://www.blindchicken.com/~ali/games/puzzles.html	Online and print puzzles Not updated regularly
http://www.pro.or.jp/~fuji/java/index-eng.html	Hirofumi Fujiwara's site with links to lots of grid puzzles, though not all puzzles generate pictures

Our current applet can be found at http://www.cs.rockhurst.edu/~cigas/visuallogic.

8. CONCLUSION

This paper has introduced the use of visual logic puzzles for teaching topics in discrete mathematics. It described one type of puzzle in detail and showed how to incorporate it into teaching logic and proofs. Furthermore, the strategies needed to solve these puzzles lend themselves well to teaching algorithms and algorithmic thinking. Additionally, it described an applet that can automate some mechanical tasks while providing a framework for studying, testing, and even coding a variety of strategies for solving these puzzles. Possible exercises and activities are mentioned, as well as links to on-line puzzle sites.

REFERENCES

DOSSEY, J.A., OTTO, A.D., SPENCE, L.E. AND VANDEN EYNDEN, C. 2002. *Discrete Mathematics, 4th edition.* Addison Wesley, Boston, MA, 88-9.

ENSLEY, D. AND CRAWLEY, W. 2005. *Discrete Mathematics: Mathematical Reasoning with Puzzles, Patterns and Games*. Wiley, New York, NY.

FUJIWARA, H. 1996. Nonogram v0.5, http://www.pro.or.jp/~fuji/java/copyright-eng.html

GREENWALD, S.J. 2003. Proof–Writing. http://www.mathsci.appstate.edu/~sjg/class/4710/sampleproofs.html.

LOCK, P.F., AND STRUTHERS, A.A. 1999. Using the game Minesweeper to introduce students to proofs. *Abstracts of Papers Presented to the American Mathematical Society* 20(1), 189.

ROSEN, K.H. 2003. *Discrete Mathematics and Its Applications, 5th edition*. McGraw-Hill, New York, 2003, 247-8.

Received March 2005; accepted August 2005