Weavable Color Patterns

Not being a mathematician, I am not obligated to complicate my explanations by excessive mathematical rigor.

— Petr Beckmann, *The History of Pi*

Introduction

Suppose you see a color pattern that strikes your fancy and think "That would make a great scarf; time to weave!".

How do you get a draft? In fact, is it even possible?

In the context of loom-controlled weaving (as opposed to, say, tapestry weaving) there are many patterns that can't be woven. In fact, most can't. We'll explore the problem of determining if a color pattern can be woven as the perpendicular interlacement of two sets of parallel threads.

The Problem

We can consider a rectangular color pattern as a grid of colored cells. If the color pattern is an image, the cells might be single pixels.

In a loom-controlled weave, every cell in the grid corresponds to a point of interlacement between a vertical (warp) thread and horizontal (weft) thread. Therefore, either the warp thread or the weft thread must be the color of the cell.

To simplify the description that follows, we'll use letters to stand for colors. See pages 8 and 9 for equivalent color patterns. Columns correspond to warp threads and rows correspond to weft threads. In order for the grid to be weavable, the columns and rows must be labeled in a way that the label for every square is its column label or its row label — "satisfied". Figure 1 shows an example grid.

	c_1	c_2	c_3
r_1	A	B	C
r_2	C	B	A

Figure 1. A Labeled Grid

We can assign column and row labels as follows. Starting with square (c_1, r_1), either c_1 or r_1 must be A. Arbitrarily pick c_1 to be A. This forces r_2 to be C. Figure 2 shows the labeling to this point.

		A		
		c_1	c_2	c_3
	r_1	A	B	C
C	r_2	C	B	A

Figure 2. First Labeling Step

We now see that c_3 must be A. This requires r_1 to be C and hence c_2 must be B. Figure 3 shows the final labeling.

		A	B	A
		c_1	c_2	c_3
C	r_1	A	B	C
C	r_2	C	B	A

Figure 3. The Final Labeling

So far, so good. But what about the grid shown in Figure 4?

	c_1	c_2	c_3
r_1	A	B	C
r_2	C	A	B

Figure 4. Another Grid

We can start as we did before, assigning A to c_1. This forces r_2 to be C, which in turn forces c_2 and c_3 to be A and B, respectively, as shown in Figure 5.

		A	A	B
		c_1	c_2	c_3
	r_1	A	B	C
C	r_2	C	A	B

Figure 5. First Step in Labeling

Now we're stuck: r_1 cannot be both B and C. If we start anywhere else and try any other combination of labelings, we find it's not possible to satisfy the grid: The pattern cannot be woven.

Note that if a larger pattern contains such a subpattern, the larger pattern cannot be woven either. Furthermore, the rows and columns do not

June 25, 2000; last revised August 1, 2004

have to be adjacent. The pattern shown in Figure 6 is equivalent to the pattern shown in Figure 5 as far as weavability is concerned.

	c_1		c_2			c_3
r_1	A	?	B	?	?	C
	?	?	?	?	?	?
	?	?	?	?	?	?
r_2	C	?	A	?	?	B

Figure 6. Separated Rows and Columns

This is a very small pattern by weaving standards and it has only three colors. What then of more colors and larger patterns?

The Number of Colors

Suppose a pattern has k colors. For $k = 2$, all patterns can be woven — simply assign one color to all columns (warp threads) and the other color to all rows (weft threads) and pick one or the other depending on the color at every intersection.

We've illustrated by example that for $k = 3$, there are some patterns that cannot be woven. For larger k there is a more fundamental problem. If a pattern has m columns and n rows, there are only $m + n$ colors available. If k is greater than $m + n$, then the pattern can't be woven at all. Thus, there are 2×3 patterns that can't be woven for this reason. See Figure 7.

	c_1	c_2	c_3
r_1	A	B	C
r_2	D	E	F

Figure 7. A Pattern with Too Many Colors

For what follows, we'll assume $k \leq m + n$.

Approaches to Solving the Problem

There are several possible ways the problem might be solved.

One way would be to try assigning the color of every cell to the columns and rows in all possible ways. This clearly is hopelessly time consuming except for tiny patterns.

Two colleagues produced viable methods and wrote programs to determine if a color image is weavable. One method recognizes the problem as an instance of the 2-satisifiabilty (2SAT) problem, for which there is a known algorithm [1]. The other solution is heuristic in nature.

We'll describe the heuristic solution here for several reasons:

- It's original as far as we know.
- It's interesting.
- It's fast for most patterns.
- It illustrates an approach that is worth considering for other problems.

The Heuristic Solution

Heuristics

A word about heuristics is in order, since they often are misunderstood. Heuristics use insights into the nature of a problem and intelligent guesses to build a solution method tailored to the problem.

Using heuristics doesn't mean wild guessing or proceeding blindly, just hoping to find a solution. Nor need a heuristic solution give incorrect answers, although proving a heuristic method is correct and terminates — and hence is an algorithm — may be difficult.

Heuristics can be used in many ways. For the problem here, one possibility would be look for a fast way to reject a pattern because it contains an unsolvable subpattern (such as the ones shown earlier). Of course, the absence of a known unweavable subpattern does not prove the whole pattern is weavable — so that problem would still exist.

Checking for special cases such as this one often takes more time on average than it saves. Since it's difficult — even impractical — to analyze the effects of such heuristics without implementing them and doing performance testing, such heuristics should be viewed with skepticism.

A good heuristic method relies on understanding the nature of the problem and, if possible, breaking the problem down into smaller, more tractable, subproblems.

Insights into Color Weavability

For the color weavability problem, the following observations are particularly useful.

- If a row or column is all one color, that color can be assigned to the corresponding row or column without affecting the rest of the problem. Hence such rows and columns can be eliminated from further consideration.
- Duplicate rows and columns can be eliminated for the same reason.
- The pattern can be rotated without changing the problem; in this sense, there is no difference between rows and columns.
- Rows can be interchanged (rearranged) without changing the problem, and the same is true of columns.

To get ideas for the heuristic approach to the problem, we can look at small subpatterns and see what implications they have for a pattern as a whole.

Two-colored patterns aren't particularly interesting, since all can be satisfied.

There are only two distinct three-colored 2×2 patterns; all others are equivalent to these by rotation or row and column interchange. See Figures 8 and 9.

	c_1	c_2
r_1	A	B
r_2	C	A

Figure 8. Three-Colored 2×2 Pattern One

	c_1	c_2
r_1	A	A
r_2	B	C

Figure 9. Three-Colored 2×2 Pattern Two

The pattern in Figure 8 imposes some constraints on any larger pattern in which it is embedded: c_1 must be A or C, c_2 must be B or A, and similarly for the two rows.

For the pattern in Figure 9, however, c_1, c_2, and r_2 are not constrained but r_1 is completely determined. It must be A for the entire pattern in which this subpattern is embedded.

This particular subpattern turns out to provide a sufficient basis for a heuristic solution; no others need be considered. We'll call this AA/BC pattern the *forcing pattern*.

A Program

What follows is a sketch of a program that implements the heuristic solution. The complete program is available on the Web [2].

Data Structures

The representation used for the pattern data is crucial. The main data structure is a vector that is used for both rows and columns.

A vector has several components, including:

an index of the row or column

a label differentiating rows and columns

a list of colors in cells

an identification of being "active" or not

An active vector is one still to be assigned a color. All vectors are active initially.

Program Structure

The program starts by reading an image file for the pattern and initializing data.

Next, duplicate rows and columns, as well as solid-colored vectors, are marked inactive. This may reduce the problem size significantly, especially for patterns with symmetries.

The main loop in the program then iterates over the pattern, developing constraints and setting colors determined by forcing patterns.

If at any time the pattern can be completely solved by simple means (see below), the problem is solved. Otherwise, all 2×2 subpatterns are examined for instances of the forcing pattern.

If a forcing pattern is found, the colors it forces are set and the loop continues. Since the cells of a forcing pattern need not be adjacent, all possible combinations of rows and columns are examined for forcing patterns.

When there are no more instances of the forcing pattern, an attempt is made solve the pattern by simple means. If this succeeds, the pattern is solved. If it fails, the pattern cannot be solved.

A pattern has a simple solution if one of the following applies:

1. The pattern is $1 \times n$ (or, equivalently, $n \times 1$) or 2×2, for which there are obvious solutions. See Figures 10 and 11.

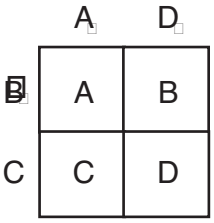

Figure 10. A 2 × 2 Pattern

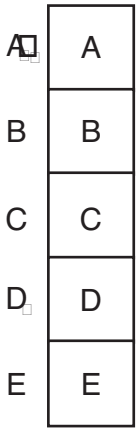

Figure 11. A 1 × 5 Pattern

2. The pattern is solid colored except for a diagonal or part of one. Again, a solution is simple. See Figure 12 for an example.

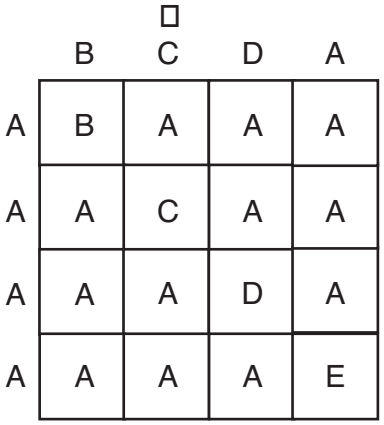

Figure 12. A 4 × 4 Diagonal Pattern

3. It can be solved by setting the color of a vector to one possibility, chosen arbitrarily, then setting colors of other vectors this forces, and continuing until all vectors have been assigned colors.

Output

On completion, the program writes a line indicating whether or not the pattern could be solved and produces lists of the row and column colors. An enlarged version of the pattern then is displayed in a window with row and column color assignments along the top, bottom, and sides. If the pattern could not be solved, the colors just reflect the program state at termination. Figure 13 shows a solved color pattern.

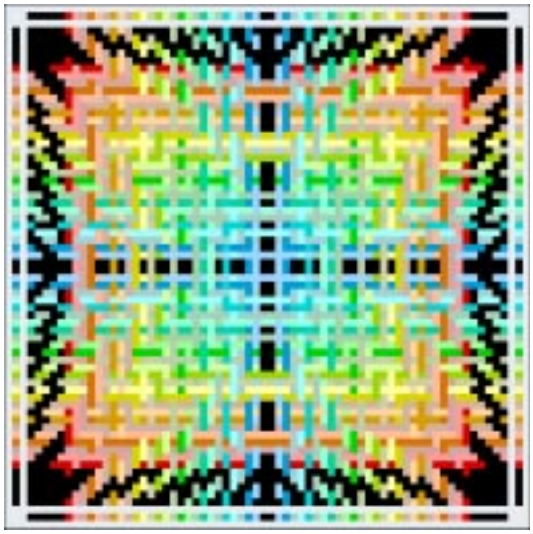

Figure 13. A Solved Pattern

Sketch of a Proof

A formal proof that the method described above is correct and terminates with a result — and hence is an algorithm — would require a formalism and a lengthy and not particularly illuminating argument.

In the spirit of the quotation at the beginning of this article, we'll just provide a sketch of a proof that, we hope, will provide some insight.

Consider what remains after eliminating duplicate rows and columns, solid-colored rows and columns, and applying all the forcing patterns.

If the remaining pattern is $1 \times n$, 2×2, or diagonal, the solution is trivial as shown earlier. Otherwise there is a 3×2 (or, equivalently, 2×3) or larger pattern containing no AA/BC forcing pattern.

If there are no rows or columns with duplicate colors, then the pattern is insoluble: a 3×2 color pattern with no duplicate color in one row or column is insoluble, as is every larger pattern of which it is a part.

The other possibility is that there is a AA/AB pattern There also may be AA/BB patterns, but there must be at least one AA/AB pattern, for otherwise the AA/BB pattern would identify two identical rows and columns, but they were eliminated earlier.

Given an AA/AB subpattern, there are only two possibilities that lead to a solution: The pattern has only two colors or it is a diagonal pattern.

This exhausts the possible patterns. Every original pattern eventually reduces to one of these cases, so the procedure terminates with a definite answer.

Getting a Draft

What remains is to use the results of a solution to create a draft — threading and treadling sequences and a tie-up.

From the color assignments for columns and rows we can get a drawdown by looking at the color of each point of intersection. Then from this we can get a draft.

For every cell in the pattern, there are three possibilities for a drawdown:

1. The corresponding row and column colors are the same, in which case either the warp or weft thread can be on top.
2. The column color is the same as the color of the point, in which case the warp thread is on top.
3. The row color is the same as the color of the point, in which case the weft thread is on top.

The first case, an option point, presents a problem — how to choose? The choice potentially is important, because it can affect the length of floats and the loom resources required.

For many patterns that might be candidates for weaving, the number of option points is huge. For the pattern shown in Figure 13, 256 of the 4,096 points are option points. So there are 2^{256} possible drafts.

It's clearly hopeless to explore even a small fraction of possible drafts that result from making different decisions at option points.

The program that creates a draft [3] provides four ways of handling option points:

choose the warp or weft at random

always chose the warp

always chose the weft

chose the warp and weft alternately

Trying each of the four methods generally gives an idea of how important the method used is and which is best. Figure 14 shows a warp-choice draft for the pattern shown Figure 13.

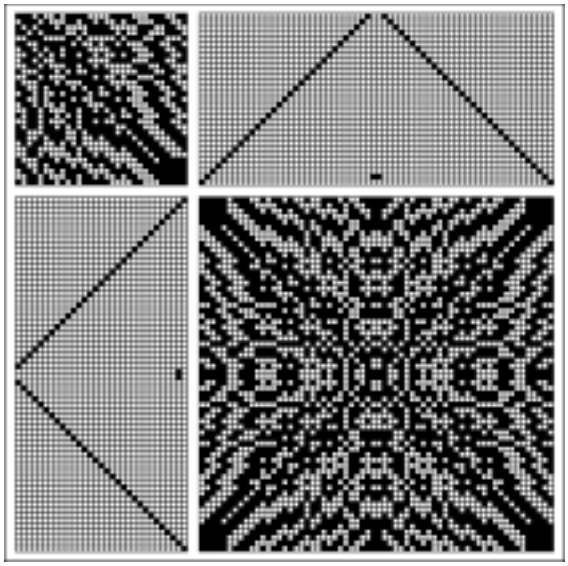

Figure 14. Warp-Choice Draft

The effects on float lengths of the method of making decisions at option points are shown in Figures 15 through 18.

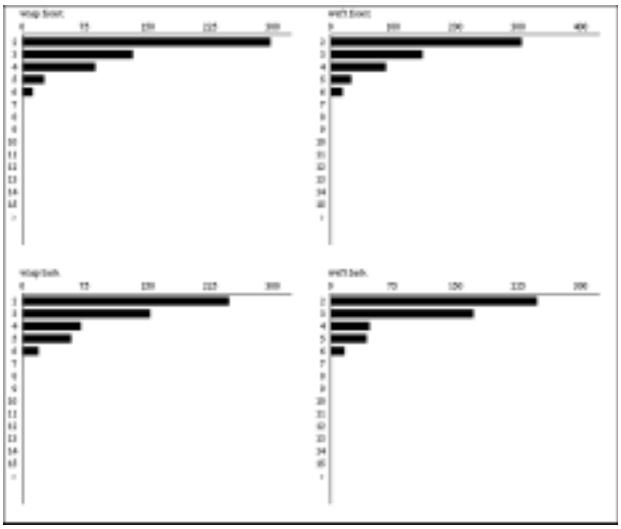

Figure 15. Random-Choice Floats

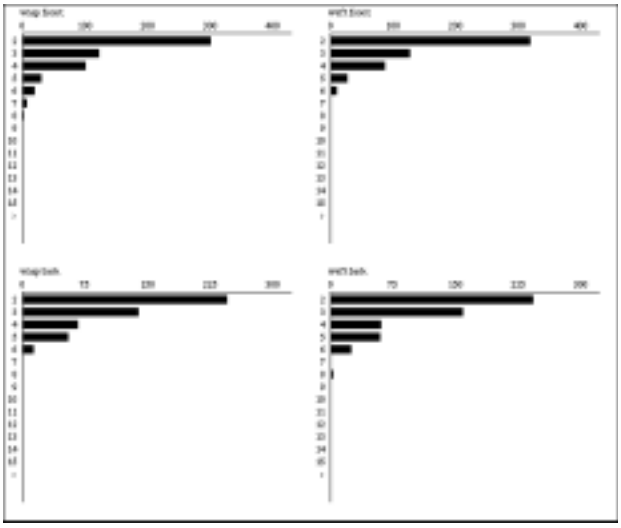

Figure 16. Warp-Choice Floats

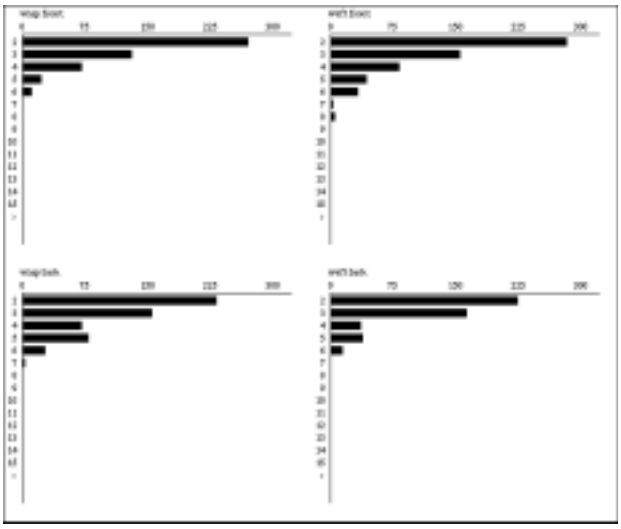

Figure 17. Weft-Choice Floats

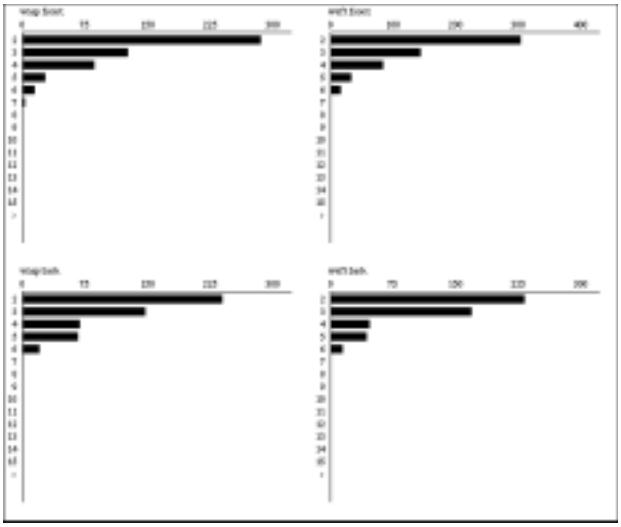

Figure 18. Alternating-Choice Floats

Another, often more important, consideration is the number of shafts and treadles the draft requires. The warp-choice draft shown in Figure 14 requires 31 shafts and 31 treadles. The weft-choice draft, shown in Figure 19, requires only 16 shafts and 16 treadles.

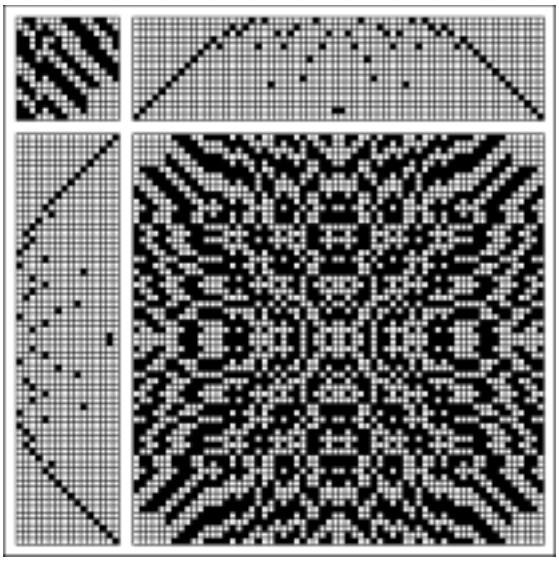

Figure 19. Weft-Choice Draft

More strikingly, the random-choice draft requires 56 shafts and 52 treadles, while the alternating choice draft requires 62 shafts and 31 treadles, making them out of the question for actual weaving.

Open Questions

The heuristic method only gives one set of colors for a solution. In some cases there may be other color assignments that satisfy the pattern, and they, in turn, might give solutions with different float structures and loom requirements. The number of alternative color assignments may be impossibly large and preclude systematic searching. The exploration of alternative solutions under the guidance of a sophisticated user might be worthwhile.

A question of more practical concern is what changes might make an unweavable pattern weavable. One possibility is to examine the effect of changes in color in forcing patterns.

Another question that could be studied is the determination of weavable subpatterns within a larger unweavable pattern.

The method described here also could be applied to the design of weavable color patterns.

Acknowledgments

Will Evans wrote the program for the 2SAT solution. Gregg Townsend developed the heuristic method, wrote the program, and supplied the proof.

References

1. *Introduction to Algorithms: A Creative Approach*, Udi Manber, Addison-Wesley, 1989.

2. *Analysis Program*, Gregg M. Townsend, 2000:
http://www.cs.arizona.edu/patterns/weaving/CAP/unravel.icn

3. *Color Drawup Program*, Ralph E. Griswold, 2000:
http://www.cs.arizona.edu/patterns/weaving/CAP/colorup.icn

Ralph E. Griswold
Department of Computer Science
The University of Arizona
Tucson, Arizona

Color Figures

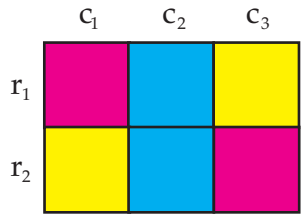

Figure 1. A Labeled Grid

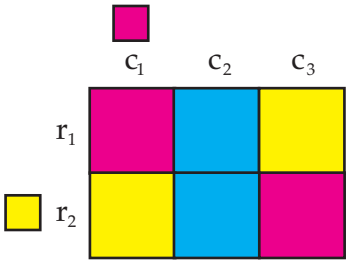

Figure 2. First Labeling Step

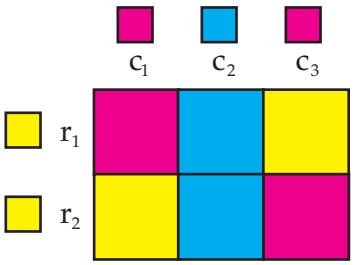

Figure 3. The Final Labeling

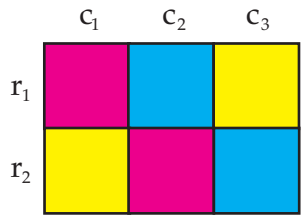

Figure 4. Another Grid

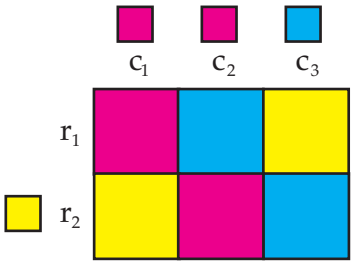

Figure 5. First Step in Labeling

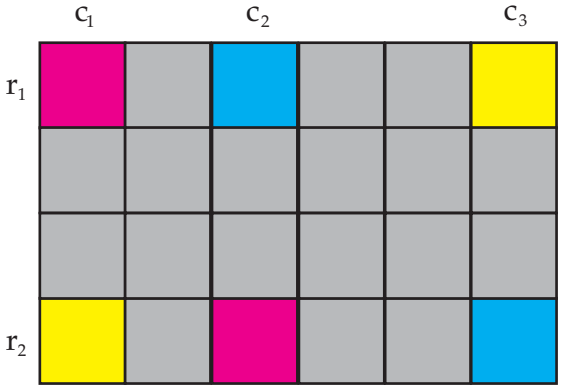

Figure 6. Separated Rows and Columns

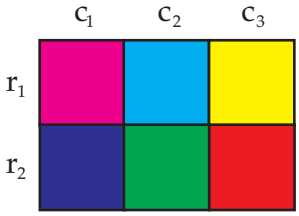

Figure 7. A Pattern with Too Many Colors

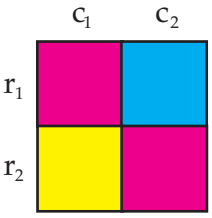

Figure 8. Three-Colored 2×2 Pattern One

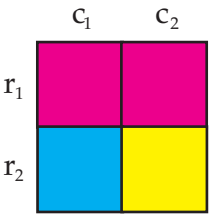

Figure 9. Three-Colored 2×2 Pattern Two

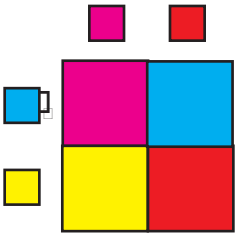

Figure 10. A 2× 2 Pattern

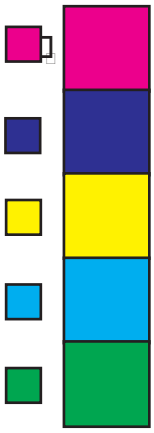

Figure 11. A 1× *n* pattern

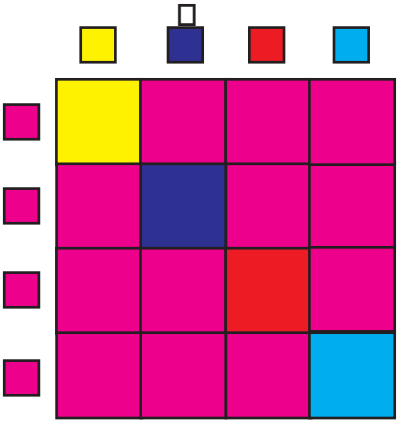

Figure 12. A 4×4 Diagonal Pattern

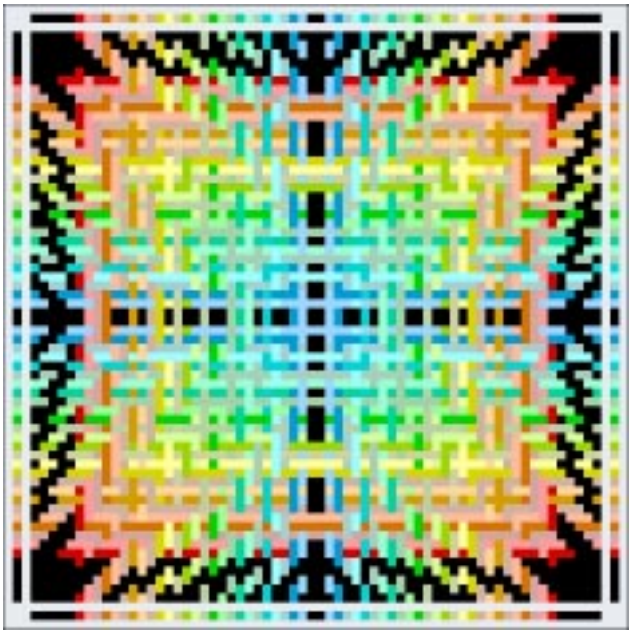

Figure 13. Solved Pattern