The Shattered Urn: Rebuilding Objects with Linear Algebra

John Burkardt

Department of Mathematics
University of South Carolina

13 November 2018

The Shattered Urn: Rebuilding Objects with Linear Algebra

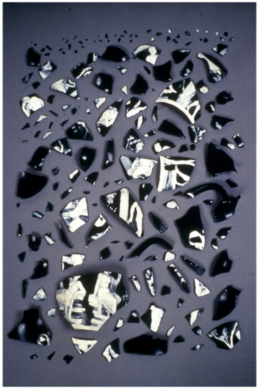
SIAM Graduate Colloquium,
Mathematics Department
University of South Carolina
...
LeConte 412
4:30-5:30, 13 November 2018

References:

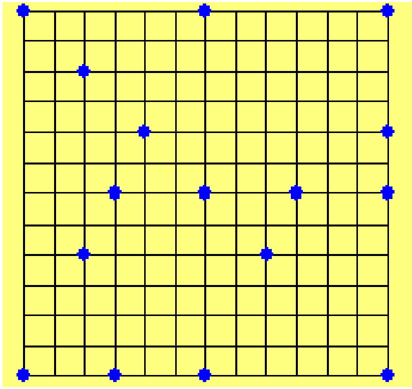
http://people.math.sc.edu/burkardt/presentations/polyominoes_2018_sc.pdf

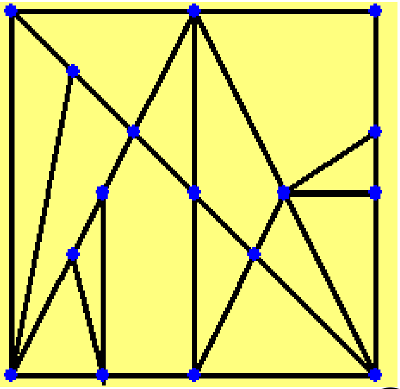
Marcus Garvie, John Burkardt,
A New Mathematical Model for Tiling Finite Regions of the Plane with Polyominoes,
submitted.

The Portland Vase 25AD - A Reconstruction Problem



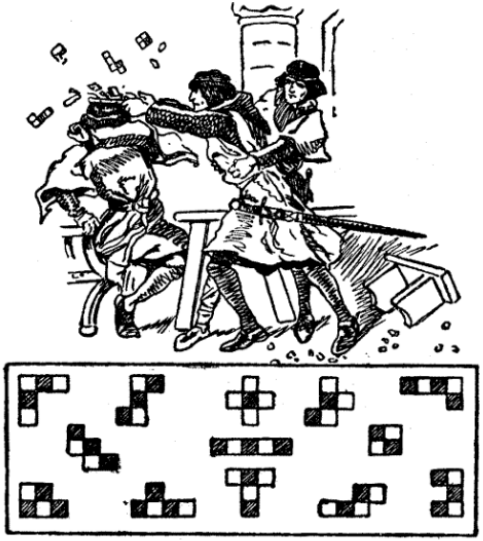
Ostomachion - Archimedes, 287-212 BC



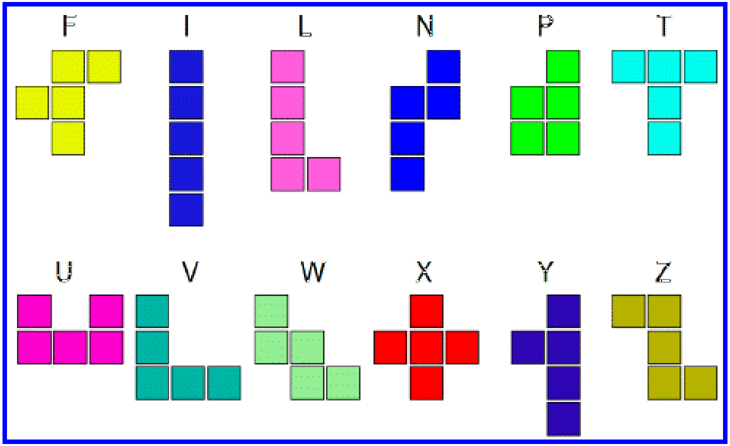


Tangrams - Song Dynasty, 960-1279

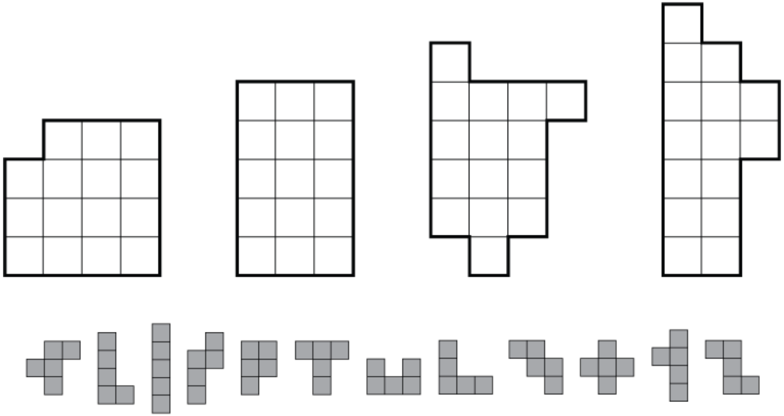
Dudeney - The Broken Chessboard (1919)



Golomb - Polyominoes (1965)

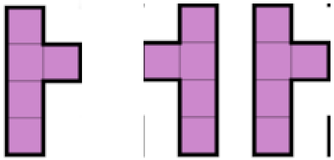


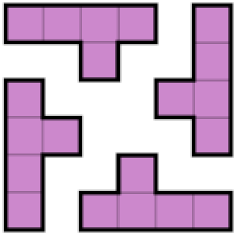
Puzzle Statement

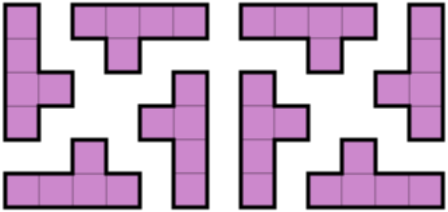


Use each pentomino exactly once. Each can be rotated or reflected.

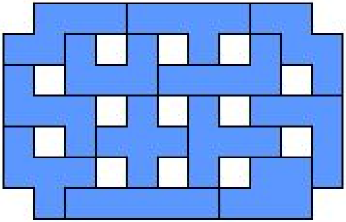
Tile Variations: Reflections, Rotations, All Orientations

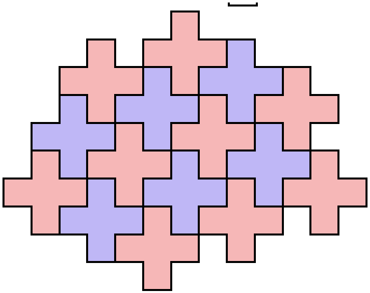






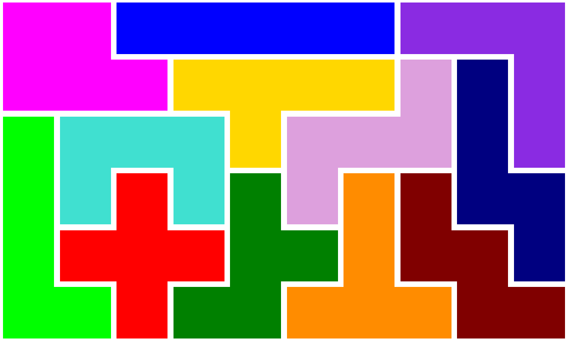
Puzzle Variations: Holes or Infinite Regions



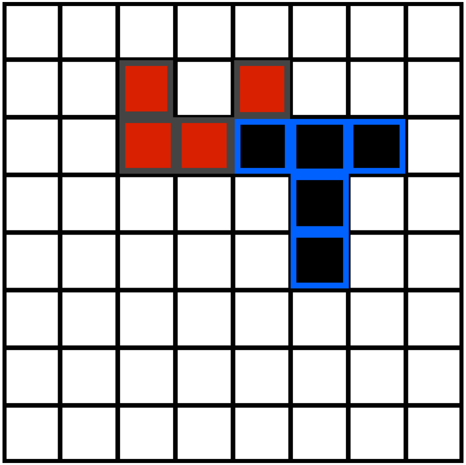


Is It Math?

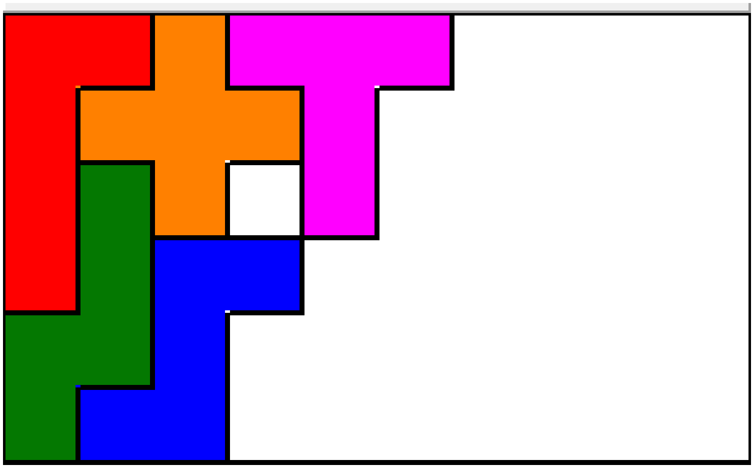
"One can guess that there are several tilings of a 6×10 *rectangle using the twelve pentominoes. However, one might not predict just how many there are. An exhaustive computer search has found that there are 2339 such tilings. These questions make nice puzzles, but are* **not the kind of interesting mathematical problem** *that we are looking for."*
"Tilings" - Federico Ardila, Richard Stanley



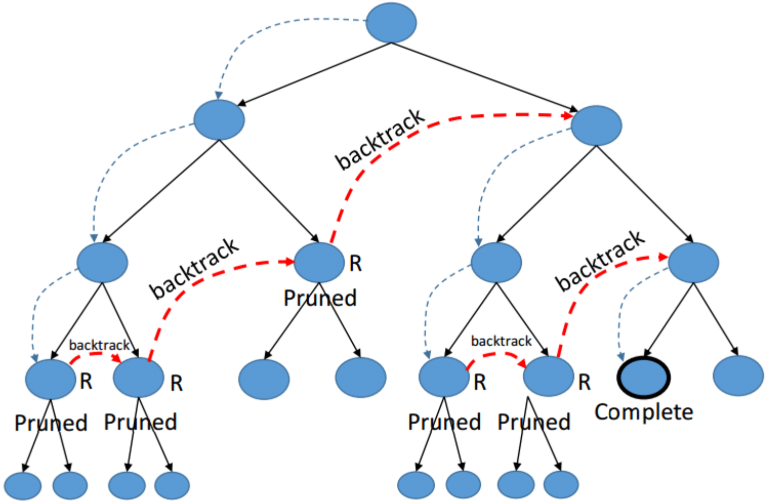
Is it “Simple” Computer Science?



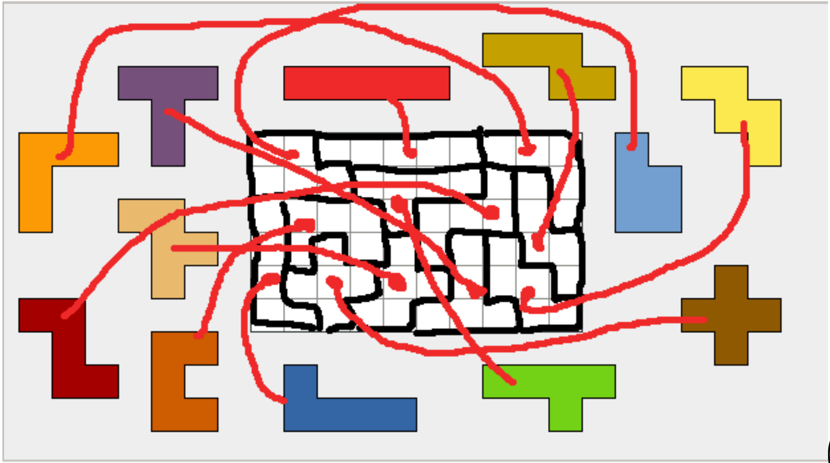
Sequential Solution: Backtracking



Is it "Clever" Computer Science?



Simultaneous Solution: Linear Algebra?



Exact Cover

Knuth: *"Tiling is a version of the exact cover problem."*

Select rows of a 0/1 matrix so every column has a 1:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Solution:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Column Sums:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Exact Cover Transposed

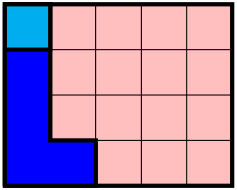
Transposed problem: select **columns** of A so every **row** has a 1:

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

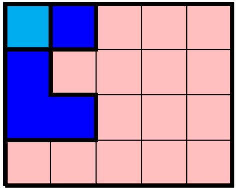
Vector x selects columns 1, 4, and 5, solving $Ax = b$:

$$\mathrm{A(:,[1,4,5])} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathrm{x} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \quad \mathrm{b} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

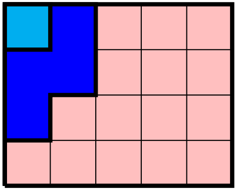
Equations: A Cell Is Covered Once By Some Tile



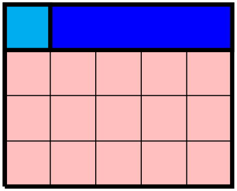
L



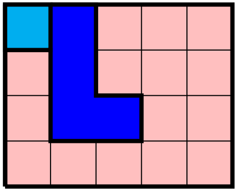
U



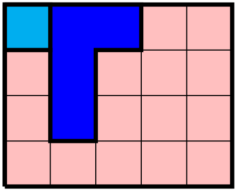
P



I

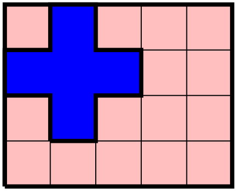


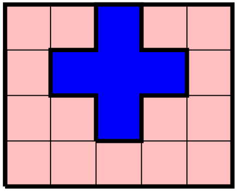
z

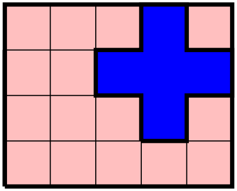


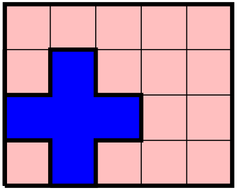
T

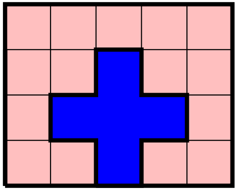
Variables: X Pentomino Example

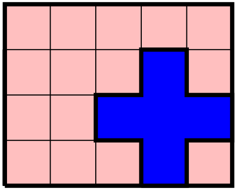

v1


v3


v5


v2


v4


v6

Constraints:

Depending on the problem and how we think about it, we may need to add some constraints, which will be either additional linear equations, or linear inequalities. Thus, for the 4×5 tiling problem just mentioned, in which the X pentomino has 6 possible placements:

Use the X pentomino exactly twice:

$$v_1 + v_2 + v_3 + v_4 + v_5 + v_6 = 2$$

Use the X pentomino no more than twice:

$$v_1 + v_2 + v_3 + v_4 + v_5 + v_6 \leq 2$$

Use a total of 4 pentominoes of any kind:

$$v_1 + v_2 + \ldots + v_n = 4$$

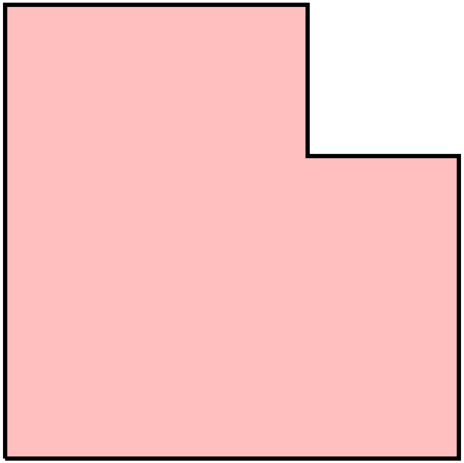
Every pentomino placement must be used once or not at all:

$$v_i \in \{0, 1\}$$

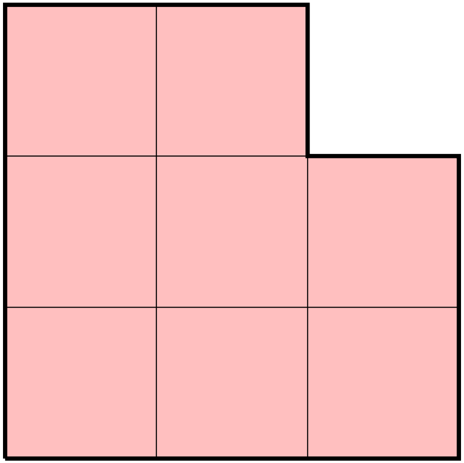
Linear System Logic

$$\left(\begin{array}{ccccccc} & V1 & V2 & V3 & V4 & V5 & V6 \\ C1 & 0 & 1 & 0 & 1 & 0 & 0 \\ C2 & 0 & 0 & 1 & 0 & 1 & 0 \\ C3 & 1 & 0 & 1 & 0 & 0 & 0 \\ C4 & 0 & 1 & 0 & 1 & 0 & 1 \\ C5 & 1 & 0 & 0 & 0 & 0 & 1 \\ C6 & 1 & 0 & 1 & 0 & 0 & 0 \\ C7 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right) \left(\begin{array}{c} use \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{array}\right) = \left(\begin{array}{c} cover? \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{array}\right)$$

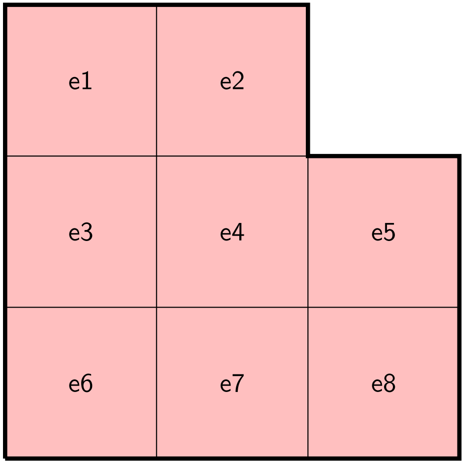
The Reid Polygon to Be Tiled



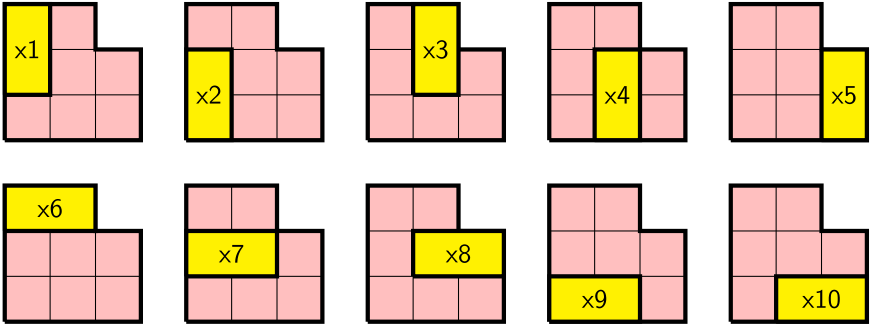
The Reid Polygon Cells



The Reid Equations: $e_1 : e_8$



The Reid Variables $x_1 : x_{10}$



We start to see how the equations and variables combine:

$$x_1 + x_6 = 1 \text{ Cell 1 must be covered once}$$
$$x_3 + x_6 = 1 \text{ Cell 2 must be covered once}$$
$$x_1 + x_2 + x_7 = 1 \text{ Cell 3 must be covered once}$$
$$\ldots \quad \ldots$$

The Reid Equations: 8 Equations, 10 Unknowns

$$
\begin{array}{lcccccccccccc}
 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & & b \\
e_1: & x_1 & & & & & +x_6 & & & & & = & 1 \\
e_2: & & & +x_3 & & & +x_6 & & & & & = & 1 \\
e_3: & x_1 & +x_2 & & & & & +x_7 & & & & = & 1 \\
e_4: & & & x_3 & +x_4 & & & +x_7 & +x_8 & & & = & 1 \\
e_5: & & & & & x_5 & & & +x_8 & & & = & 1 \\
e_6: & & x_2 & & & & & & & +x_9 & & = & 1 \\
e_7: & & & & x_4 & & & & & +x_9 & +x_{10} & = & 1 \\
e_8: & & & & & x_5 & & & & & +x_{10} & = & 1
\end{array}
$$

The Reid Linear System: 8 Equations, 10 Unknowns

$$A\,x = b$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Reduced Row Echelon Form of Reid Linear System

Notice that variables 7, 9 and 10 are free!

$$\begin{array}{lcccccccccccc}
 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & & b \\
e_1: & 1 & & & & & & 1 & & -1 & & = & 0 \\
e_2: & & 1 & & & & & & & 1 & & = & 1 \\
e_3: & & & 1 & & & & 1 & & -1 & & = & 0 \\
e_4: & & & & 1 & & & & & 1 & 1 & = & 1 \\
e_5: & & & & & 1 & & & & & 1 & = & 1 \\
e_6: & & & & & & 1 & -1 & & 1 & & = & 0 \\
e_7: & & & & & & & & 1 & & 1 & = & 0 \\
e_8: & & & & & & & & & & & = & 0
\end{array}$$

Drop Zero Row, Add Degrees of Freedom

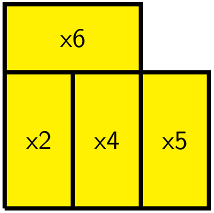
We add placeholder equations for variables 7, 9 and 10.

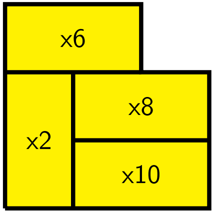
$$\begin{array}{rcccccccccccc}
 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & & b \\
e_1: & 1 & & & & & & 1 & & -1 & & = & 0 \\
e_2: & & 1 & & & & & & & 1 & & = & 1 \\
e_3: & & & 1 & & & & 1 & & -1 & & = & 0 \\
e_4: & & & & 1 & & & & & 1 & 1 & = & 1 \\
e_5: & & & & & 1 & & & & & 1 & = & 1 \\
e_6: & & & & & & 1 & -1 & & 1 & & = & 0 \\
f_1: & & & & & & & 1 & & & & = & ? \\
e_7: & & & & & & & & 1 & & 1 & = & 0 \\
f_2: & & & & & & & & & 1 & & = & ? \\
f_3: & & & & & & & & & & 1 & = & ?
\end{array}$$

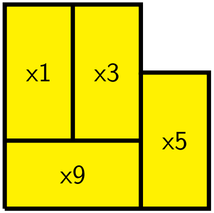
Solutions for 8 Binary Right Hand Sides

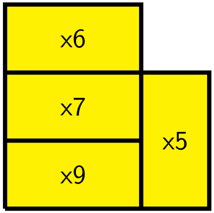
$$\begin{array}{r@{\;:\;}rrrrrrrr}
x_1 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 \\
x_2 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
x_3 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 \\
x_4 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \\
x_5 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
x_6 & 1 & 1 & 0 & 0 & 2 & 2 & 1 & 1 \\
x_7 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
x_8 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
x_9 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
x_{10} & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}$$

The Reid Tilings (Labeled)









A Universal Algorithm for Polyomino Tiling?

1. Number the cells in the polygon (equations);
2. Number every placement of a polyomino in the polygon (variables);
3. Construct the linear system;
4. Solve the linear system.

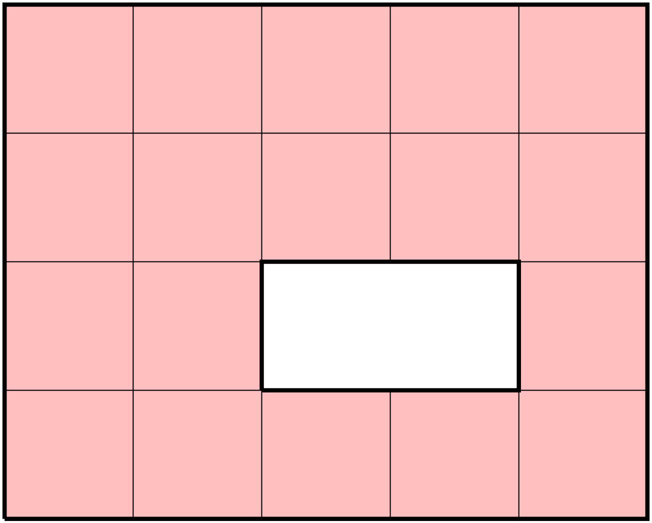
Formally OK, but what's hard here? What could go wrong?

A More Realistic Algorithm for Polyomino Tiling

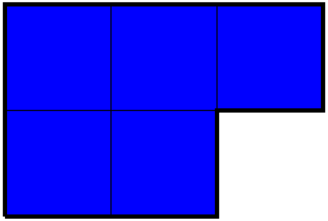
1. Number the cells in the polygon (equations);
2. If multiple polyomino shapes, add constraint equations;
3. Generate every rotation or reflection of each polyomino;
4. Number every placement of a polyomino in the polygon (variables);
5. Construct the (rectangular) linear system;
6. Expand to a square system, including degrees of freedom;
7. Generate every possible right hand side;
8. Exact solution required (integer system, rational answers!);
9. Only accept "binary" solutions.
10. Count, plot, analyze solutions.

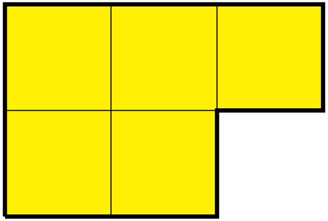
In some cases, **thousands** *of solutions may be found.*

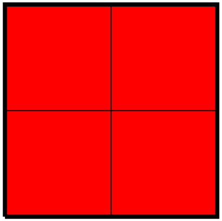
4x5 Example: Region R

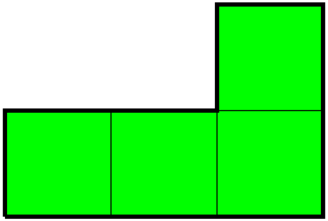


4x5 Example: Tiles P









MATLAB Programs

Go to **http://people.sc.fsu.edu/∼jburkardt**

Choose the MATLAB examples

Look for **polyomino_monohedral_example_reid**

Many more polyomino codes available.

4x5 Example: Discussion of coding

```
%
%  Step 1: Define Region:
%
  r_m = 4;
  r_n = 5;
  r_shape = ones ( r_m, r_n );
  r_shape(3,3:4) = 0;

  r_shape

  1 1 1 1 1
  1 1 1 1 1
  1 1 0 0 1
  1 1 1 1 1
```

4x5 Example: Discussion of coding

```
%
%  Step 2: Define Polyomino Shapes:
%
  p_num = 3;
  p_shapes = zeros ( r_m, r_n, p_num );

  p_shapes(1,1:3,1) = 1;
  p_shapes(2,1:2,1) = 1;

  p_shapes(1:2,1:2,2) = 1;

  p_shapes(1,3,3) = 1;
  p_shapes(2,1:3,3)= 1;
```

4x5 Example: Discussion of coding

```
%
%  Step 3: Check Polyomino Shapes:
%
  p_shapes(1:2,1:3,1)
  1 1 1
  1 1 0

  p_shapes(1:2,1:2,2)
  1 1
  1 1

  p_shapes(1:2,1:3,3)
  0 0 1
  1 1 1
%
%  Step 4: Indicate multiplicity.
%
  d = [2; 1, 1];
```

4x5 Example: Discussion of coding

```
%
%  Step 5: Construct linear system A*x=b:
%
  [ a, b, parent ] = polyomino_multihedral_matrix ( ...
    r_shape, p_num, p_shapes, d );
  [ m, n ] = size ( a );
%
%  Step 6: Write LP file.
%
  polyomino_lp_write ( '4x5.lp', 'Test', m, n, a, b );
%
%  Step 7: Get Reduced Row Echelon Form
%
  ab = [ a, b ];
  [ ab_rref, det ] = r8mat_rref ( m, n + 1, ab );
```

4x5 Example: Discussion of coding

```
%
%  Step 8: Solve for binary solutions.
%
  nz = sum ( d );
  [ x_num, x ] = r8mat_rref_solve_binary_nz ( m, n, ...
    nz, a_rref, b_rref );
%
%  Step 9: Print array of solutions.
%
  for i = 1 : n
    for j = 1 : x_num
      fprintf ( 1, '%2d', x(i,j) );
    end
    fprintf ( 1, '\n' );
  end
```

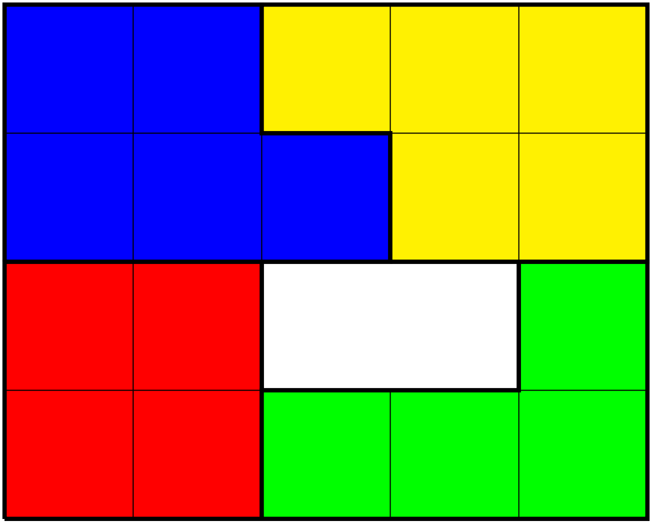
4x5 Example: Discussion of coding

```
%
%  Step 10: Plot each solution as a tiling.
%
  for j = 1 : x_num
    label = sprintf ( '  Tiling based on solution %d', j
    polyomino_multihedral_tiling_print ( r_shape, p_num,
      p_shapes, d, x(:,j), label );
  end
```

4x5 Example: Run the code

- RREF system has 23 rows and 62 columns;
- Augmented system has 42 degrees of freedom;
- ALL binary right hand sides is 2^{42}, on the order of a **trillion**;
- Only check binary RHS with at most 4 degrees of freedom set to 1: $1+42+42*41/2+42*41*40/6+42*41*40*39/24=124,314$;
- Generated and solved all 123,314 right hand sides, found 4 binary solutions in less than 7 seconds.

4x5 Example: Tiling # 1 of 4



Obstacle: Inaccurate & Slow A*x=b Solvers

- The row-reduced echelon form (RREF) is very sensitive to roundoff.
- We can't rely on MATLAB's rref(A) command, (real arithmetic).
- A "hand-made" integer arithmetic code, can handle small problems.
- Tiling problems of moderate size have many degrees of freedom "D".
- The number of possible solutions we will need to check rises like 2^D.
- To solve interesting problems, need accurate, efficient integer solver;

Accurate & Efficient A*x=b Solvers!

Solving underdetermined integer problems A*x=b turns out to be an activity of enormous interest, especially in the linear programming community, in which the problem can include the request to optimize a corresponding cost function $\phi(x)$.

MATLAB Optimization Toolbox includes **optimvar()**.

Fast and efficient solvers are freely available: CPLEX, Gurobi, SCIP.

Moreover, the linear programming community uses a simple **LP** file format to describe such problems. So our task can be simplified:

- Set up the problem, write it to an LP file;
- Call an appropriate integer linear programming solver;
- Retrieve the solution, count, plot, analyze it;

Reid Example: LP file “reid.lp”

```
  \ The Reid example

Maximize
  Obj: 0
Subject to
 x1 + x6 = 1
 x1 + x7 = 1
 x2 + x6 + x8 = 1
 x2 + x3 + x7 + x9 = 1
 x3 + x10 = 1
 x4 + x8 = 1
 x4 + x5 + x9 = 1
 x5 + x10 = 1
 x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10 = 4
Binary
  x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
End
```

Reid Example: Solving with CPLEX

```
(1) CPLEX> set mip pool absgap 0.0
(2) CPLEX> set mip pool intensity 4
(3) CPLEX> set mip limits populate 10
(4) CPLEX> set mip pool capacity 10
(5) CPLEX> set output writelevel 1
(6) CPLEX> read reid.lp
(7) CPLEX> populate
(8) CPLEX> write reid.xml all
```

1. Set the absolute gap for the solution pool to zero;
2. Aggressively seek all solutions;
3. Upper bound on solutions sought;
4. Upper bound on solutions stored;
5. Write all solutions to file;
6. Read LP file;
7. Seek solutions;
8. Write solution file.

Reid Example: Retrieving Solutions

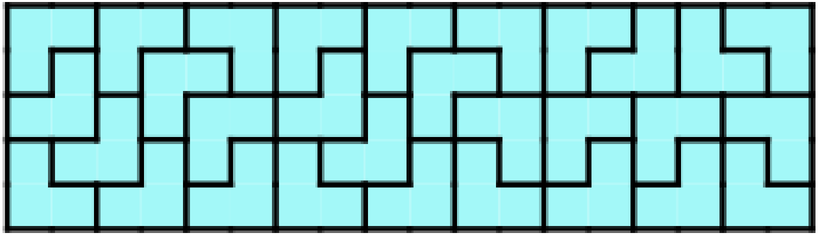
Each package has its own way of reporting the solutions. CPLEX output uses the XML format. If we use CPLEX to solve the Reid problem, then each of the 4 solutions will be stored, along with much more information.

Here is what one Reid solution looks like:

```
<CPLEXSolution>
  ...
<variables>
 <variable name="x2" index="4" value="1"/>
 <variable name="x6" index="14" value="1"/>
 <variable name="x4" index="8" value="1"/>
 <variable name="x5" index="12" value="1"/>
</variables>
</CPLEXSolution>
```

With help from an XML reader, (thanks, Wouter Falkena!) we can retrieve each solution.

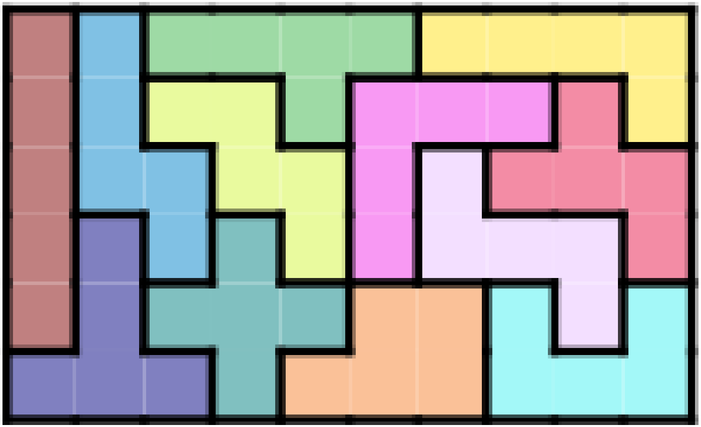
Tiing by Multiple Copies of a Single Polyomino



1 triomino, 4 orientations, 90+1 equations, 272 variables

1,168,512 solutions computed by CPLEX in 3.8 minutes.

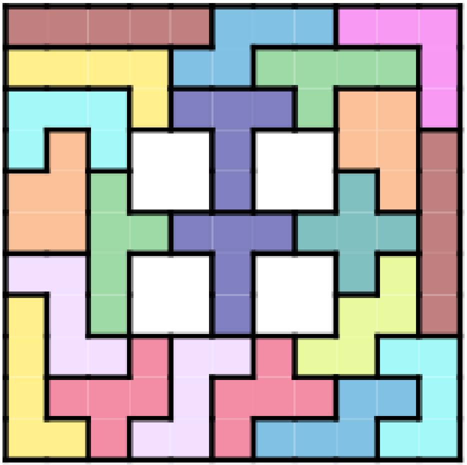
Tiing by One Copy of Each Pentomino



12 pentominoes, 1/2/4/8 orients, 60+12+1 equations, 2056 variables

9,356 solutions computed by CPLEX in 7.3 minutes.

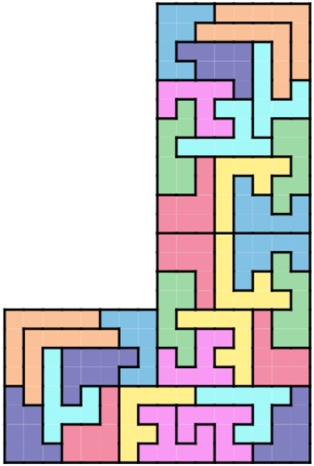
Tiling a Region with Holes



12 pentominoes, 1 or 2 copies, 118 equations, 2,619 variables

8 (equivalent) solutions computed by CPLEX in 0.4 seconds.

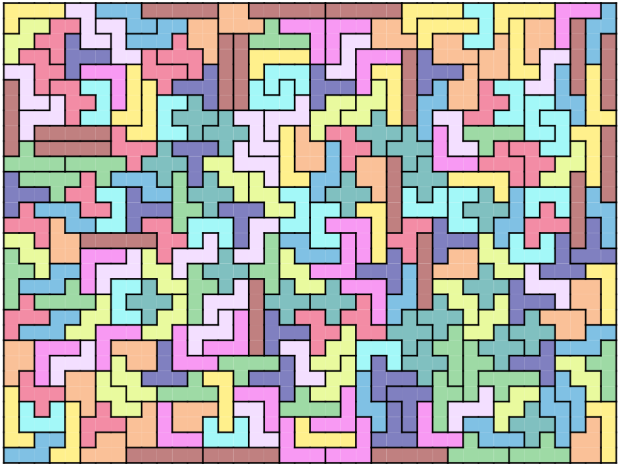
Tiling a Nonrectangular Region



8 octominoes, 4 copies each, 265 equations, 9,878 variables

1 solution computed by CPLEX in 13 minutes.

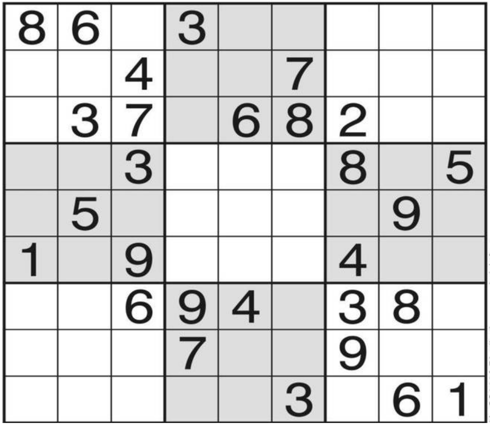
A Large Problem



12 pentominoes, 20 copies each, 1,213 equations, 67,396 variables

8 (equivalent) solutions computed by CPLEX in 9.5 minutes.

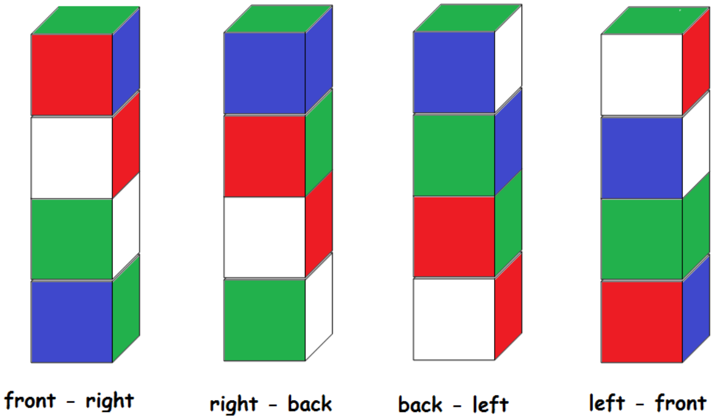
Sudoku



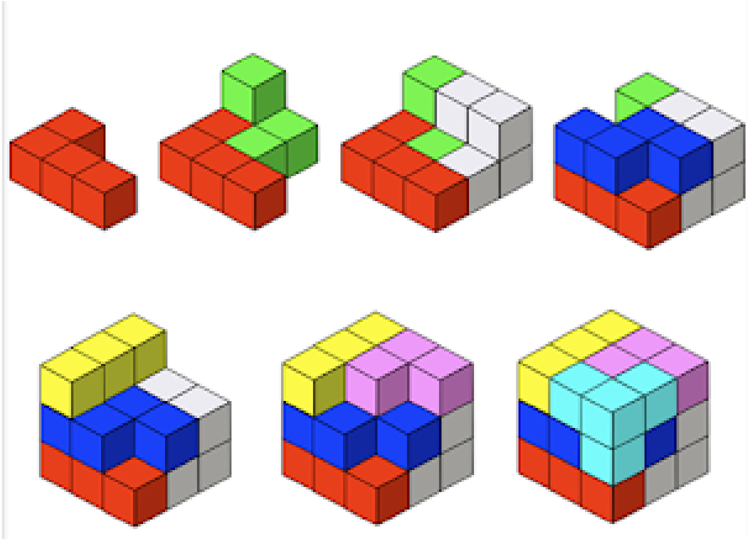
8	6		3					
		4			7			
	3	7		6	8	2		
		3				8		5
	5						9	
1		9				4		
		6	9	4		3	8	
			7			9		
					3		6	1

Instant Insanity

Stack four cubes so all four sides show one of each color...

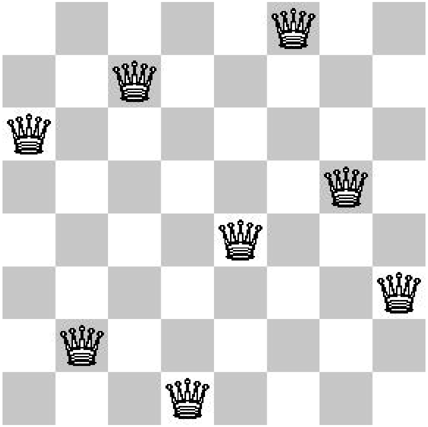


Piet Hein's Soma Cube

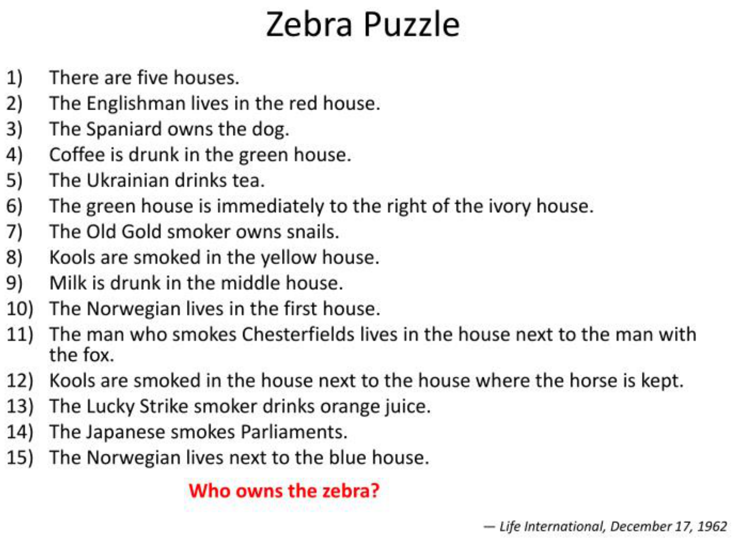


N Queens

Place N queens on an $N \times N$ chessboard so no pair are attacking.



Who Owns the Zebra?



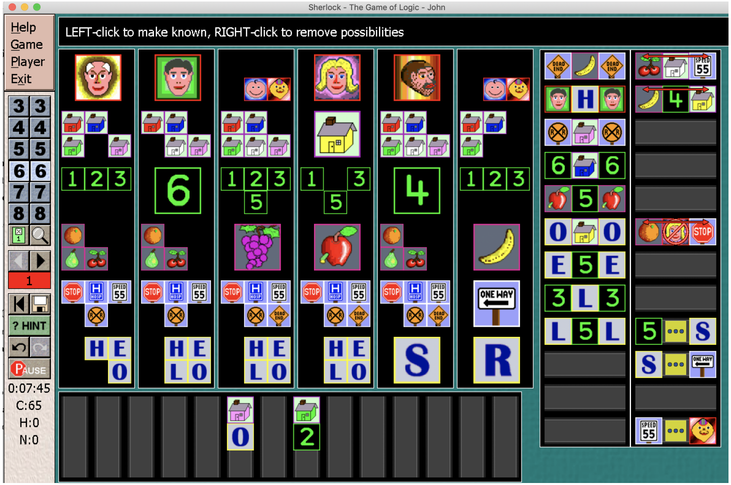
Zebra Puzzle

1) There are five houses.
2) The Englishman lives in the red house.
3) The Spaniard owns the dog.
4) Coffee is drunk in the green house.
5) The Ukrainian drinks tea.
6) The green house is immediately to the right of the ivory house.
7) The Old Gold smoker owns snails.
8) Kools are smoked in the yellow house.
9) Milk is drunk in the middle house.
10) The Norwegian lives in the first house.
11) The man who smokes Chesterfields lives in the house next to the man with the fox.
12) Kools are smoked in the house next to the house where the horse is kept.
13) The Lucky Strike smoker drinks orange juice.
14) The Japanese smokes Parliaments.
15) The Norwegian lives next to the blue house.

Who owns the zebra?

— *Life International, December 17, 1962*

Sherlock:


Sherlock - The Game of Logic - John
Help
Game
Player
Exit
LEFT-click to make known, RIGHT-click to remove possibilities
? HINT
PAUSE
0:07:45
C:65
H:0
N:0

http://www.kaser.com

Conclusion: Rebuilding with Linear Algebra

To solve the tiling problem, we must see the underlying cell structure.

- Region cells correspond to **equations**.
- Tile cells correspond to **variables**.
- Together, they form an **underdetermined linear system**.

By manipulating free variables, we catalog all possible solutions.

And that's how we restore the shattered urn!

The Portland Vase: COMMENTS

The Portland Vase is a unique object, made from glass 2000 years ago.

It may have been dug up and reconstructed around 1600.

It became a property of the Portland family in 1784; the base was broken in 1810, and after repair, it was place in the British Museum for "safekeeping".

In 1845, a drunken visitor smashed the vase into pieces. A restoration was attemped, although 37 small pieces were left over.

Another restoration was done in 1948, and again most recently in 1988.

This is a practical example of the reconstruction problem, in which a known object is to be put together using some or all of the pieces supplied.

Ostomachion: COMMENTS

The Ostomachion, or Stomachion, is one of the earliest dissection or tiling puzzles known. It is attributed to Archimedes. It was only discovered as part of the Archimedes Codex, a manuscript that had later been scraped clean of ink and reused as a prayer book.

The name Ostomachion can be broken down as osto+machion, or bone-battle and suggests that the puzzle was manufactured out of pieces of bone.

The Ostomachion is not a good test case for the algebraic reformulation of a tiling problem, since the individual puzzle pieces cannot be formed by combining copies of a single, simpler cell shape.

Tangram: COMMENTS

The Tangram puzzle is believed to have originated in China, sometime during the Song dynasty [960-1279].

Tangram patterns are called in China "qiqiaoban" meaning *seven boards of skill.*

Tangrams came to the West around 1815.

Notice that all seven shapes can be decomposed into congruent triangles, which will mean that tangram puzzles can be handled by our algebraic approach.

Dudeney: COMMENTS

H R Dudeney was one of the earliest popularizers of mathematical, geometrical and logical puzzles.

He contributed short puzzles to English magazines and newspapers; these were popular enough that he was able to later publish several collections of them, including "The Canterbury Puzzles", which were framed as being posed by the pilgrims of Chaucer's "Canterbury Tales".

The broken chessboard is one such puzzle, and is perhaps the first appearance of the puzzle tiles known as "pentominoes". The 64 squares of a chessboard are to be covered by 12 different pieces, each made of 5 squares, plus a single 2x2 square.

These days, the puzzle is often posed with a chessboard with a missing 2x2 hole, so that only the 12 pentominoes are to be used.

Golomb: COMMENTS

In 1965, Solomon Golomb published a book *"Polyominoes"* describing a family of tiling puzzles related to shapes created by joining squares together. Since two squares made a "domino", he named the shapes made from 3, 4 and 5 squares as *trominoes*, *tetraminoes* and *pentominoes*.

After Martin Gardner reviewed the book and its puzzles in his Scientific American column, polyominoes, and particularly pentominoes, became a popular puzzle type, and enlivened the mathematical study of tilings.

Puzzle Statement: COMMENTS

This is actually a single puzzle, with the user having to choose how to divide up the tiles.

Our algebraic approach can handle this, since it is not necessary for the region to be connected.

Loosely speaking: "Cover this polygonal region with these tiles."

We will always be looking at a finite polygonal region, such that both the region and the tiles are constructed of square cells;

We will allow holes in the region.

We will be assuming that the type and number of polyominoes to be used is given in advance.

Reflections, Rotations, All Orientations: COMMENTS

In most problems, the polyomino tiles may be used in any orientation. As long as we are working with tiles made from squares, this means that a particular tile may have 1, 2, 4, or even 8 distinct orientations, depending on whether we allow reflections, rotations, or both. Allowing orientations gives us more flexibility in solving a puzzle, but it also can greatly increase the number of possible solutions we need to check.

If we start with 12 pentominoes in a particular orientation, we can actually generate:

- 18 distinct shapes if we are allowed reflections.
- 41 if we are only allowed rotations;
- 63 if we are allowed reflections and rotations;

Puzzle Variations: COMMENTS

In some cases, the problem of tiling the infinite plane is much easier to answer than for a specific polygonal region.

For example, it's easy to make a diagonal line of **X** pentominoes. And we can stack one diagonal after another next to each other, and we have a tiling of the infinite plane.

And it's obvious that **X** can't be used alone to tile any rectangular region.

But many regions can be created for which this question is difficult to answer.

Is It Math?: COMMENTS

There is a mathematical folk tale that Gauss, as a schoolboy, asked to sum the integers from 1 to 100, immediately said $50 * 101 = 5050$.

A mathematician is delighted by this idea, because it shows that there is a way to look at a boring, mindless problem that exposes a structure, and thus a way to answer the question rapidly, and which also suggests how to solve similar problems.

If certain tiling problems can only be solved by "exhaustive computer search", this seems to leave no avenue for human cleverness or insight, and thus might be left to computer scientists to crunch on.

But perhaps there is more math in polyomino tiling than we suspect.

Is it "Simple" Computer Science?: COMMENTS

Does an exhaustive computer search merely mean we just throw pieces on the board and see if they tile?

Some random computations, such as estimating an integral by the Monte Carlo method, do work this way.

But for the tiling problem, we have constraints that make a simple random process very unlikely to succeed:

- shape of the polygon means not all placements will work;
- we can't allow two polyominoes to overlap;
- we have to choose a particular reflection and rotation of each polyomino;
- if we don't keep track of our previous results, we may repeat some many times, never know when to stop, and never notice some solutions.

Is it "Clever" Computer Science?: COMMENTS

Many computer science algorithms are inspired by human activities that fail simply because we lack patience or perfect memory.

A computer science approach to the tiling problem simulates the process of attempting a tiling one piece at a time, that is, **as a sequential decision process**.

Every time we reach a dead end, we cancel the last decision, and back up to the previous one, and try a different choice.

This procedure, if done right, guarantees that every correct sequence of decisions will be discovered.

But the procedure itself is pretty mindless, and makes a mathematical analysis very difficult!

Sequential Solution: Backtracking: COMMENTS

Here's an example of a backtracking process in action.

The solver has just put down the purple **T** piece.

We humans can see that this is a big mistake, because we know the isolated square cannot be covered now.

Unless we make the solver smart enough to see this, it will continue adding a piece at a time until hitting a dead end where no tile can be placed at all.

It will slowly back up, checking more bad decisions, until it reaches the step where it placed the purple **T** piece.

After canceling that decision, it can proceed, and there may be hope yet for a solution.

Simultaneous Solution:Linear Algebra?:COMMENTS

The step by step sequential approach to the tiling problem is naturally appealing; after all, the alternative is to try to place all the pieces simultaneously, and how could you do that?

Well, we do that all the time! A set of linear equations is called a simultanous set of equations because, at the very same time, we want multiple things (the variables) to cooperate (satisfy multiple equations).

If we can frame the tiling problem as a system of linear equations, then we have all sorts of methods to analyze the problem, and many techniques for finding solutions.

We replace **sequential** by **simultaneous** processing.

Variables: X Pentomino Example: COMMENTS

A given cell can be covered by any of the tiles, and that tile may be in various locations; it may also be rotated or reflected.

We need a variable to represent every different tile shape, and every reflection, rotation, and translation of it.

To keep it simple, look at the **X** pentomino, and a 4 row by 5 column rectangular array of cells. This pentomino generates six variables, $v_1, v_2, \ldots, v_6$, which represent all the ways that the tile can be placed in the polygon.

The value of each variable will be 0 or 1, depending on whether the X pentomino appears in the tiling at exactly that position.

Exact Cover:COMMENTS

The exact cover is an abstract class of problems in which an object, composed of many units, is to be reconstructed out of a subset of a given set of components.

Each component covers some of the units.

The subset of components must be chosen so that each. unit is covered exactly once.

Knuth recognized that tiling was an instance of the exact cover problem.

Exact Cover Transposed: COMMENTS

Knuth describes a problem involving matrix rows.

We can transpose the matrix, and seek columns of the matrix so that there's a single 1 in every row.

Choosing the columns can be done by multiplying A by a vector x of 0's and 1's.

Now we have turned our problem into linear algebra:

Find a "binary" vector x so that $A * x = b$ where b is a vector of 1's

Equations: A Cell Is Covered Once: COMMENTS

Is the tiling problem hiding a linear system somewhere?

An equation says something has to happen a certain way.

We know that each cell of the polygon must be covered exactly once.

The tiling problem becomes a sequence of statements:

- the number of tiles covering cell 1 must be 1;
- the number of tiles covering cell 2 must be 1;
- ...
- the number of tiles covering cell M must be 1;

If these statements play the role of equations, what do we use for variables?

Variables: COMMENTS

Each polyomino can be placed on the region in many ways.

We think of the *i*th possible placement as a variable v_i.

It takes some thought to determine, for a given polyomino, all the possible translations, rotations, and reflections that represent distinct placements onto the region.

But once we have done that, we have a natural language in which to represent every possible tiling of the region.

Constraints: COMMENTS

If we are given a single polyomino of p squares, and asked to tile a region of r squares, then we expect that r is exactly divisible by p, and that the number of placements we use is exactly r/p. This is a constraint.

Moreover, thinking the ith possible placement as a variable, we expect either to use that placement or not. In other words, each variable in a tiling should have the value 0 or 1.

If we are allowed to use a given polyomino d times (or no more than d times), then several corresponding v's can be nonzero, and we must add a constraint that the sum of these v's must be d (or no more than d).

Constraint equations are easy to include in the linear system.

Constraint inequalities can be handled by linear programming software.

Linear System Logic: COMMENTS

The linear system $A * x = b$ can be read as
Can I choose tile orientations so each cell is covered once?

Note that: though:

- the matrix A is probably rectangular;
- the linear system is generally underdetermined;
- acceptable solution vectors x must be "binary";
- how do we guarantee that no tile exceeds the region?
- how do we guarantee every tile is used once, or the right number of times?

Such problems are a form of the linear programming problem, called **binary integer linear programming** or **BILP**. There is a wealth of mathematical knowledge about this problem, and powerful software to solve enormous versions of such systems (with thousands or even millions of equations.

So now we need to see how to go about looking at a typical tiling problem and seeing a set of linear equations.

The Reid Tiling Problem: COMMENTS

Consider the region R to be a 3×3 checkerboard, after removing the upper right square.

Let D be the domino tile, formed using two adjacent squares.

Use 4 copies of D, in any orientation, to cover the region.

Find all possible solutions.

Michael Reid,
The homotopy groups,
L' Enseignement Mathématique,
Volume 49, pages 123-155, 2003;

Reduced Row Echelon Form: COMMENTS

Every $m \times n$ matrix can be written in reduced row echelon form (RREF) by carrying out a version of Gauss-Jordan elimination.

A matrix is in RREF if:

1. The first nonzero entry in each row is 1;
2. The leading 1 in a given row occurs in a column to the right of the leading 1 in the previous row;
3. Rows which are entirely zero must occur last;
4. Each column with a leading 1 has no other nonzero entries;

If the right hand side is appended to the matrix, RREF can determine if the underdetermined system is consistent, and can characterize solutions.

Roundoff is a problem, even for a completely integer system, because there is division.

Reduced Row Echelon Form: COMMENTS

Row-reduced echelon form (RREF) of $[A|b]$:

- Move to row I, seeking a pivot for column J.
- If A(I,J) is zero, increase I.
- Otherwise, divide row I by A(I,J) to get a pivot of 1.
- Add a multiple of row I to all other rows, clearing column J.
- Columns with no pivot correspond to degrees of freedom.

Solutions can be constructed by assigning free variables to 0 or 1, and using the RREF to solve for the determined variables.

We can't guarantee in advance that determined variables will come out with a value of 0 or 1, but these are the only acceptable results.

4x5 Example: Discussion of Coding: COMMENTS

I probably won't have time to go through this coding example.

I include it to show how the tiling problem can be broken into a logical series of steps.

These steps can be naturally translated into a computer program in a language such as MATLAB or Python.

For large cases, however, it is really necessary to rely on an expert programmer, or a powerful and flexible integer linear programming package. This is because of issues of accuracy (no roundoff!), efficiency (avoid unneeded storage and calculations), power (do really large problems) and flexibility (handle constraints and inequalities).

Displaying Results: COMMENTS

Each solution x is just a list of the variables to be set to 1.

As a simple check, we can verify that A*x=b, that our solution solves the linear system that represents the tiling problem.

But to believe the answer, we need good plots.

- Outline the region;
- Outline each tile;
- Choose and assign colors;
- Publication quality;

We argued ("discussed") this problem for weeks! (TikZ won)

Sudoku: COMMENTS

Let x_{ijk} indicate that in row i column j we placed the digit k.

Then we have four sets of constraints:

- Every row i must include digit k exactly once (9x9 constraints);
- Every column j must include digit k exactly once (9x9 constraints);
- Every box must contain digit k exactly once (9x9 constraints);
- Every cell (i, j) must have only one nonzero k (9x9 constraints);

This results in $9 \times 9 \times 9 = 729$ variables and $4 \times 9 \times 9 = 324$ constraints.

We add to the system roughly 20 or 30 equations setting particular values of x_{ijk}, and we arrive at a system logically the same as the tiling problem.

Instant Insanity: COMMENTS

We abstractly represent the “geometry” of the problem as a 4×4 grid:

```
        Red  Green  Blue  White
        -----------------------
front |    |      |     |    |
      ------------------------
right |    |      |     |    |
      ------------------------
back  |    |      |     |    |
      ------------------------
left  |    |      |     |    |
      ------------------------
```

Each cube has 24 orientations (6 choices for top, 4 turns).

Instant Insanity: COMMENTS

As shown in the solution, the cube $\#1$, in the given orientation, partially "tiles" the region as follows:

	Red	Green	Blue	White
front	1			
right			1	
back			1	
left				1

and we just need to choose orientations for the other three cubes so that the region is tiled.

N Queens:COMMENTS

The "geometry" of the N Queens problem is also abstract.

The cells, each of which must be covered by exactly one queen, are **not** the individual squares of the chess board!

The cells we need to cover are actually collections of squares:

- the 8 rows (exactly 1 queen in each);
- the 8 columns (exactly 1 queen in each;

along with 26 "less than" constraints:

- the 13 non-unit diagonals (less than or equal to 1 queen in each);
- the 13 non-unit anti-diagonals.(less than or equal to 1 queen in each)

The variables, of course, are the 64 possible positions of a queen.

Who Owns the Zebra?:COMMENTS

Five houses, colors, cigarette brands, pets, drinks, nationalities:

- The Englishman lives in the red house;
- The Spaniard owns the dog;
- Coffee is drunk in the green house;
- The Ukrainian drinks tea;
- The green house is immediately to the right of the ivory house;
- The Old Gold smoker owns snails;
- Kools are smoked in the yellow house;
- Milk is drunk in house #3;
- The Norwegian lives in house #1;
- The Chesterfields smoker lives next to the man with the fox;
- Kools are smoked in the house next to the house with the horse;
- The Lucky Strike smoker drinks orange juice;
- The Japanese smokes Parliaments;
- The Norwegian lives next to the blue house;

Who drinks water? And who owns the zebra?

Who Owns the Zebra?:COMMENTS

Five houses, colors, cigarette brands, pets, drinks, nationalities:

- The Englishman lives in the red house;
- The Spaniard owns the dog;
- Coffee is drunk in the green house;
- The Ukrainian drinks tea;
- The green house is immediately to the right of the ivory house;
- The Old Gold smoker owns snails;
- Kools are smoked in the yellow house;
- Milk is drunk in house #3;
- The Norwegian lives in house #1;
- The Chesterfields smoker lives next to the man with the fox;
- Kools are smoked in the house next to the house with the horse;
- The Lucky Strike smoker drinks orange juice;
- The Japanese smokes Parliaments;
- The Norwegian lives next to the blue house;

Who drinks water? And who owns the zebra?

Sherlock COMMENTS

At http://www.kaser.com, Everett Kaser offers a number of puzzle programs for Windows and Mac OSX systems, with free downloadable demo versions.

Some of these puzzles can be seen to be equivalent to a tiling problem.

In the Sherlock game, a matrix is given, whose rows are people, houses, street numbers, favorite fruit, a traffic sign, and a letter.

Clues (constraints) include:

- "The Dead End sign must be next to the banana".
- "The pink house has the letter O"
- "The white house has cherry on one side, speed limit on the other"
- "The speed limit sign is to the left of the smiling man."

References:

- Federico Ardila, Richard Stanley, *Tilings*, https://arxiv.org/abs/math/0501170, 2005;
- Matthew Busche, *Solving Polyomino and Polycube Puzzles: Algorithms, Software, and Solutions*, http://www.mattbusche.org/blog/article/polycube/;
- Henry Ernest Dudeney, *The Canterbury Puzzles*, 1919;
- Wouter Falkena, **xml2struct**, MathWorks File Exchange;
- Marcus Garvie, John Burkardt, *A New Mathematical Model for Tiling Finite Regions of the Plane with Polyominoes*, submitted;
- Solomon Golomb, *Polyominoes*, Allen and Unwin, 1965;
- Donald Knuth, *Dancing Links*, Millennial Perspectives in Computer Science, pages 187-214, 2000;
- Reviel Netz, William Noel. *The Archimedes Codex*, Weidenfeld and Nicolson,2007;
- Michael Reid. *The homotopy groups*, L' Enseignement Mathématique, Volume 49, pages 123-155, 2003;