Cracking the Humpty Dumpty Case: Rebuilding Broken Objects with Linear Algebra

John Burkardt
Host: AMCUS 2022

Department of Mathematics
University of Pittsburgh at Greensburg
10:00am, Saturday
117 Cassell Hall

09 April 2022

T-puzzles by the Open Lab `https://teaching.pitt.edu/open-lab/`
https://people.sc.fsu.edu/∼jburkardt/presentations/t_puzzle_2022_upg.pdf

....

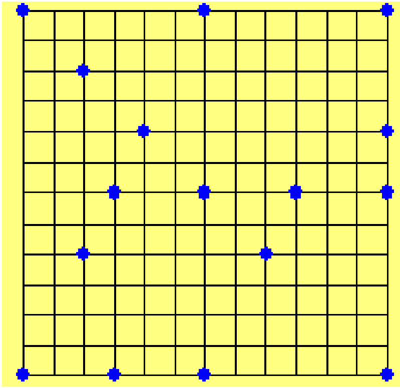
....

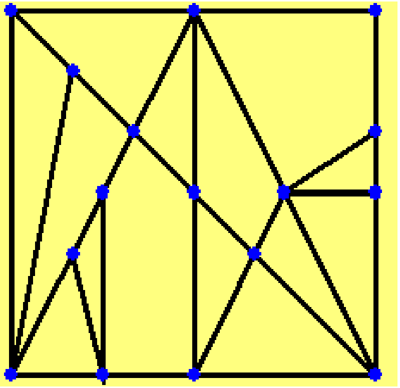
The Humpty Dumpty Problem

Humpty Dumpty sat on a wall,
Humpty Dumpty had a great fall.
All the king's horses and all the king's men
Couldn't put Humpty together again.

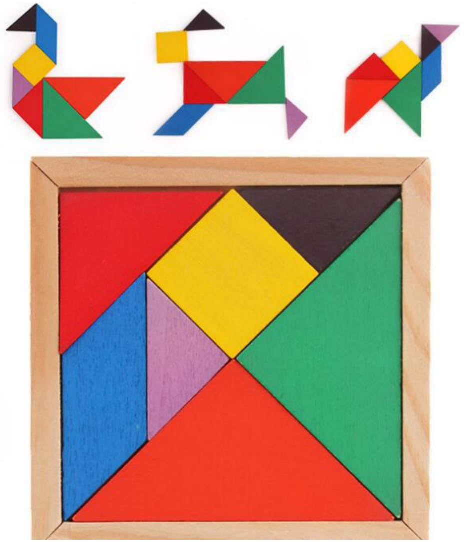
But maybe we can!

Ostomachion - (Archimedes, 287-212 BC)

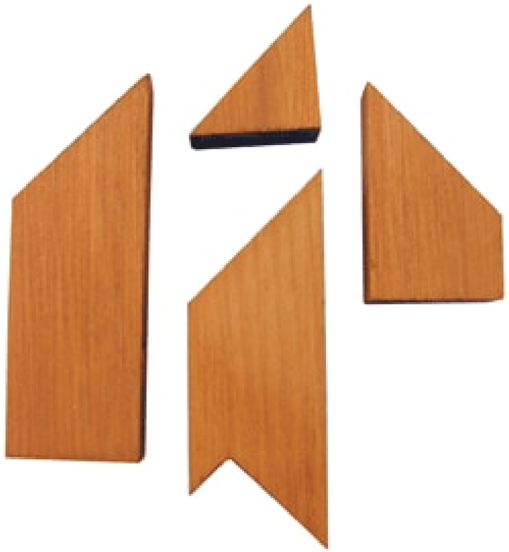




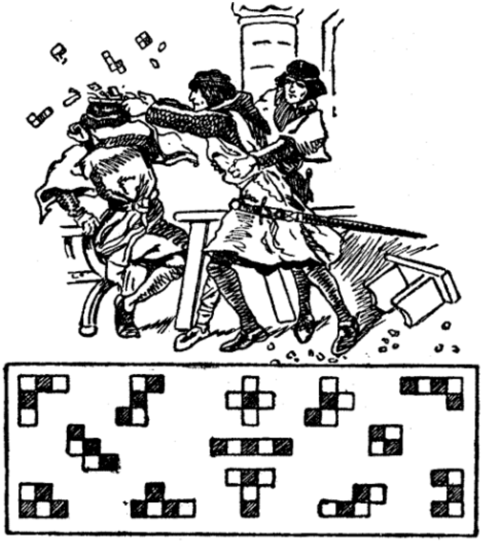
Tangrams - (Song Dynasty, 960-1279)



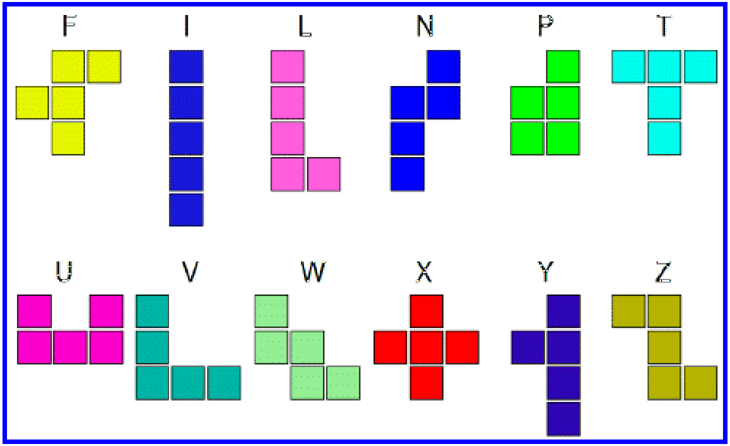
The T Puzzle - (1800)



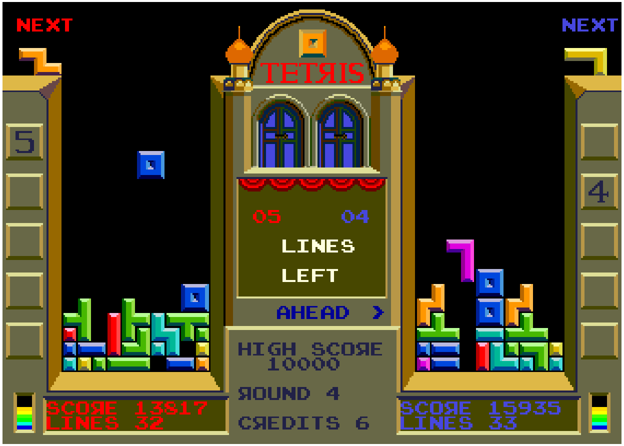
The Broken Chessboard (Dudeney, 1919)



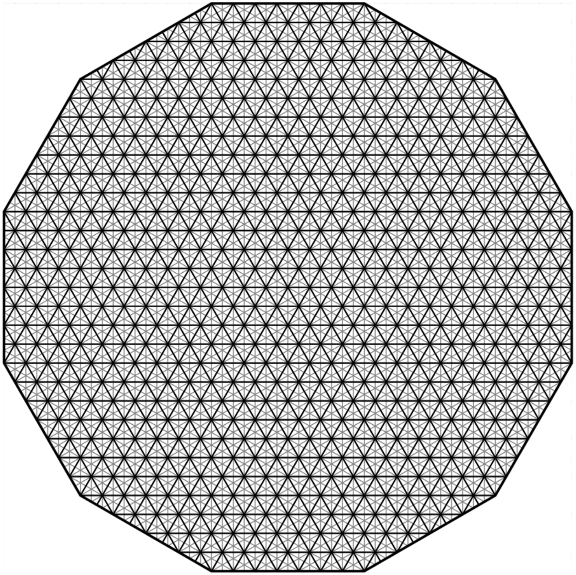
Pentominoes & Polyominoes (Golomb, 1965)



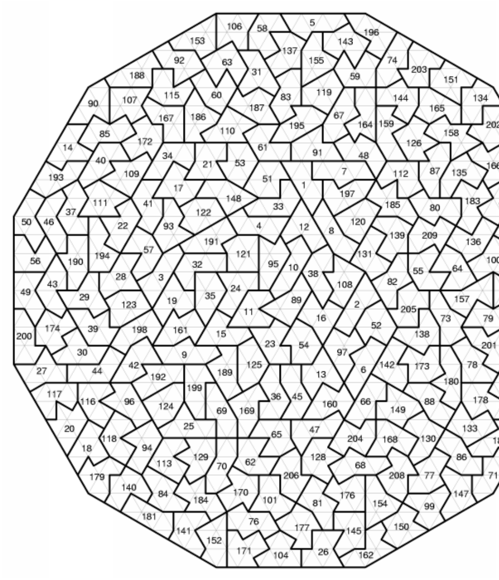
Tetris (1989)


NEXT
NEXT
TETRIS
5
4
05
04
LINES
LEFT
AHEAD
HIGH SCORE
10000
ROUND 4
CREDITS 6
SCORE 13817
LINES 32
SCORE 15935
LINES 33

Eternity - (Monckton, 1999)

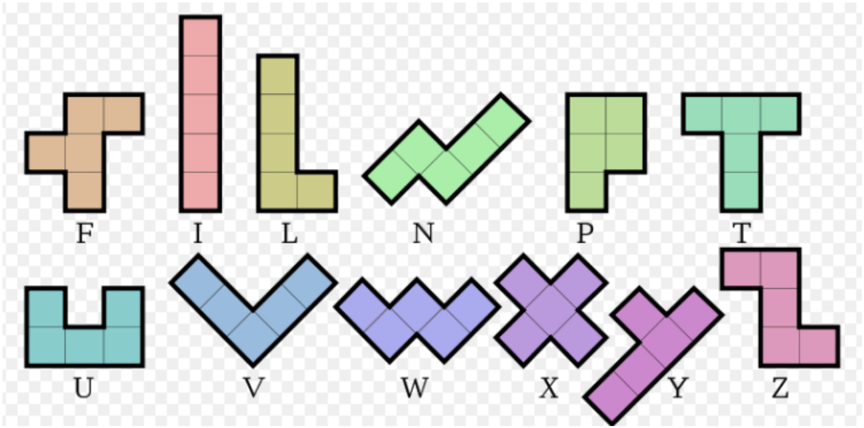


Eternity - solved for £1,000,000 prize (Selby, 2000)



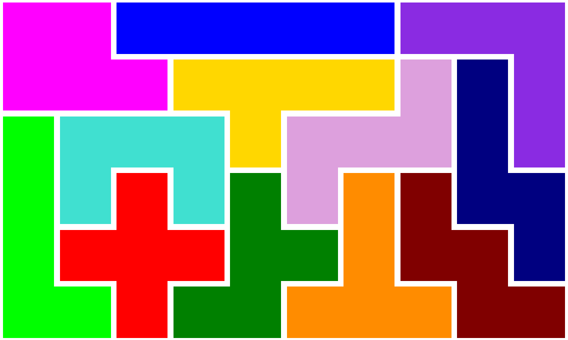
Puzzle Statement

Can we tile a 6x10 rectangle with the 12 pentominos?

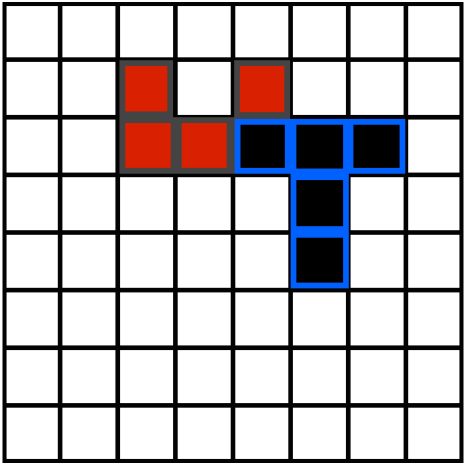


OK, But Is It Math?

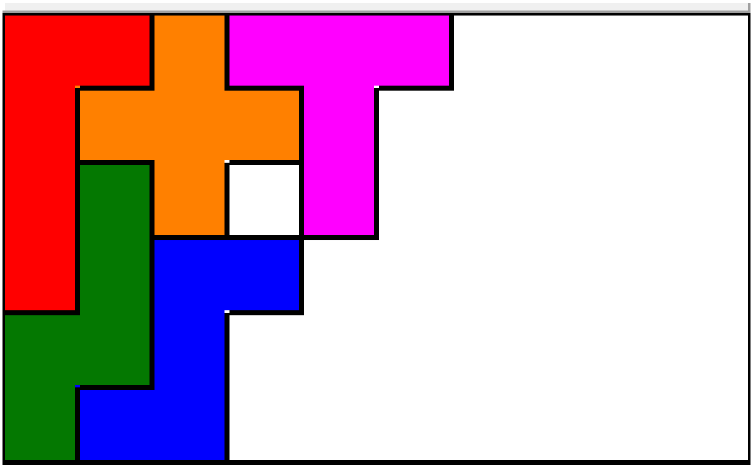
"One can guess that there are several tilings of a 6×10 *rectangle using the twelve pentominoes. However, one might not predict just how many there are. An exhaustive computer search has found that there are 2339 such tilings. These questions make nice puzzles, but are* **not the kind of interesting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



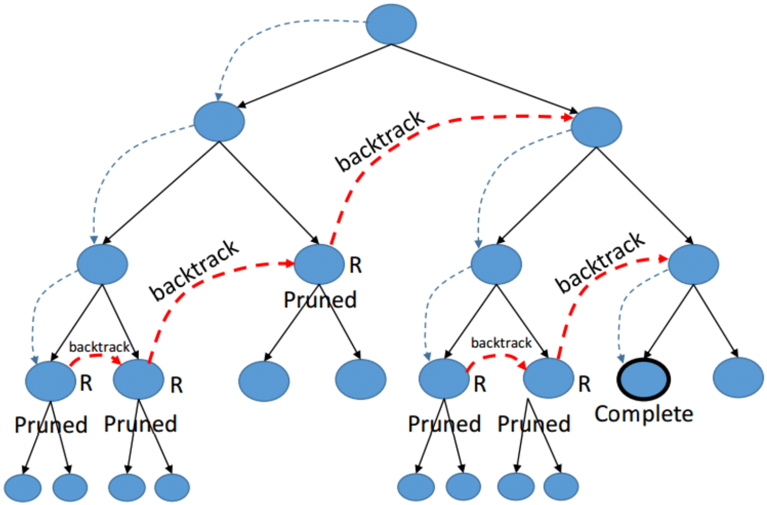
Is it “Simple” Computer Science?



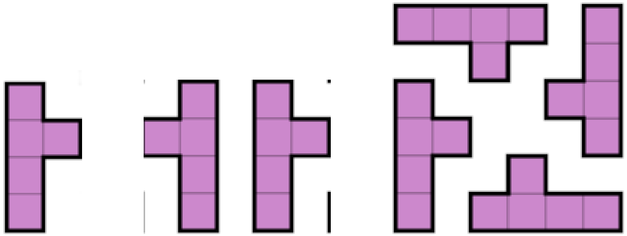
Sequential Solution: Backtracking

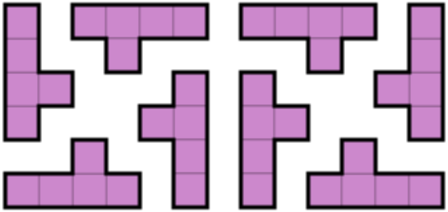


Is it "Clever" Computer Science?

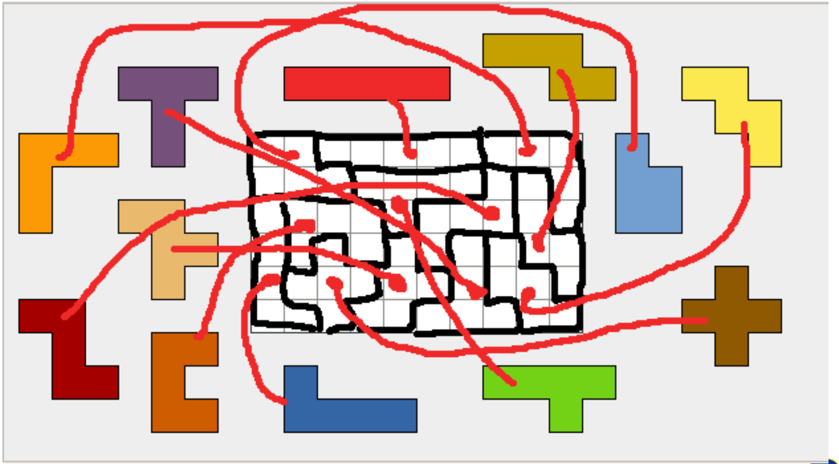


Tile Variations: Reflections, Rotations, All Orientations





Simultaneous Solution: Linear Algebra?



Linear Algebra can set multiple objects that satisfy multiple requirements.

Exact Cover

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

Select columns of a 0/1 matrix so every row has a single 1:

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

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

Exact Cover as a Linear System

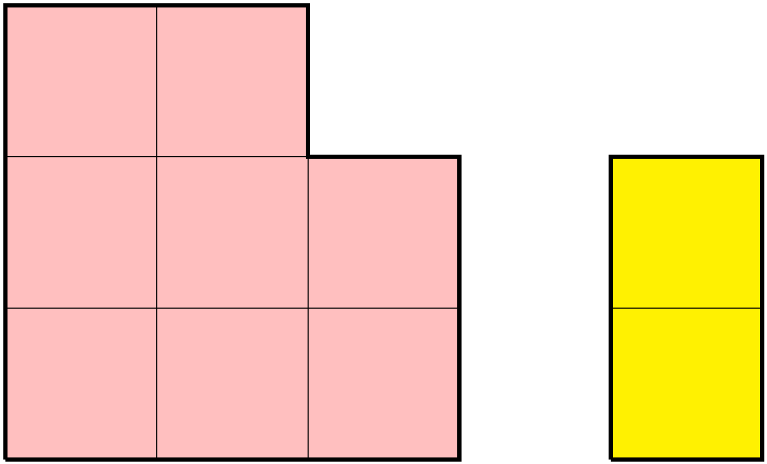
Select columns of A so every row has a single 1.

This is equivalent to
*Find x so that A*x=b where b is a vector of 1's.*

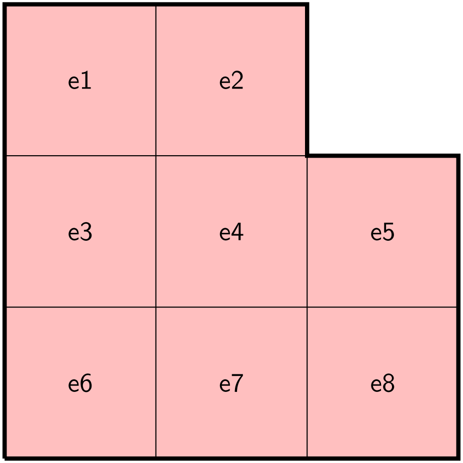
Now this is linear algebra!

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

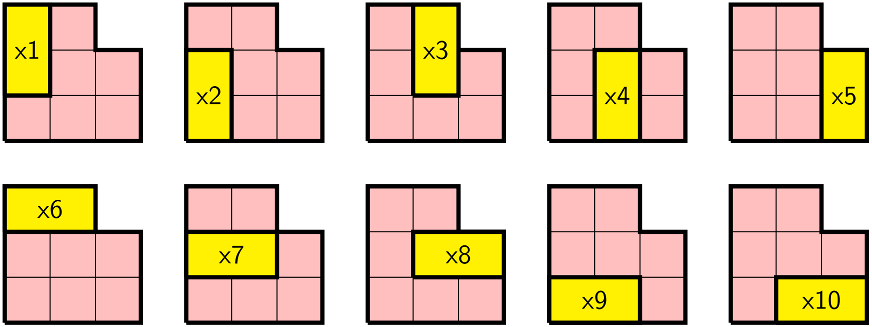
Tile the Reid Polygon with 4 Dominoes



The Reid Equations: $e_1 : e_8$



The Reid Variables $x_1 : x_{10}$



We start to see how the equations and variables combine:

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

The Reid Equations: 8 Equations, 10 Unknowns

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

The Reid Linear System: 8 Equations, 10 Unknowns

$$\mathrm{A}\ \mathrm{x} = \mathrm{b}$$

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

Reduced Row Echelon Form of Reid Linear System

Notice that variables 7, 9 and 10 are free!

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

Equation 8 disappears because once we have covered the first 7 cells, cell 8 is guaranteed to be covered.

Drop Zero Row, Add Degrees of Freedom

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

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

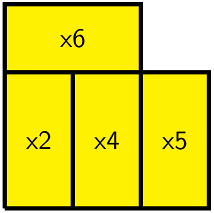
Solutions for 8 Binary Right Hand Sides

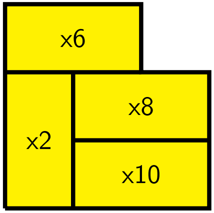
x_1 :	0	0	1	1	-1	-1	0	0
x_2 :	1	1	0	0	1	1	0	0
x_3 :	0	0	1	1	-1	-1	0	0
x_4 :	1	0	0	-1	1	0	0	-1
x_5 :	1	0	1	0	1	0	1	0
x_6 :	1	1	0	0	2	2	1	1
x_7 :	0	0	0	0	1	1	1	1
x_8 :	0	1	0	1	0	1	0	1
x_9 :	0	0	1	1	0	0	1	1
x_{10} :	0	1	0	1	0	1	0	1

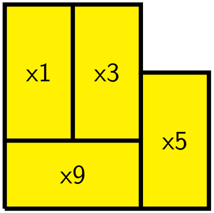
4 Acceptable Solutions (only 0 and 1 values)

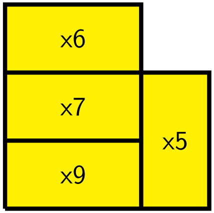
	✓	✓	✓	×	×	×	✓	×
x_1 :	0	0	1	1	-1	-1	0	0
x_2 :	1	1	0	0	1	1	0	0
x_3 :	0	0	1	1	-1	-1	0	0
x_4 :	1	0	0	-1	1	0	0	-1
x_5 :	1	0	1	0	1	0	1	0
x_6 :	1	1	0	0	2	2	1	1
x_7 :	0	0	0	0	1	1	1	1
x_8 :	0	1	0	1	0	1	0	1
x_9 :	0	0	1	1	0	0	1	1
x_{10} :	0	1	0	1	0	1	0	1

The Reid Tilings (Labeled)









Bigger Problems Need a Better Solver

The Reid linear system A*x=b was 8 equations in 10 unknowns. It was easy to write a code to reduce A and b, via reduced row-echelon form; then to deal with the free variables, and then to eliminate solutions with unacceptable values. But for larger problems, this approach won't work.

- 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 code can only handle small problems.
- Tiling regions can have hundreds of cells (equations/rows $= M$).
- Tiling problems can have tens or hundreds of tiles $= T$.
- A tile may have roughly M configurations, not even counting rotations and reflections (variables/columns $N \approx T * M$).
- The linear system may 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 solutions, count, plot, analyze;

Reid Example: LP file “reid.lp”

```
\ The Reid problem

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

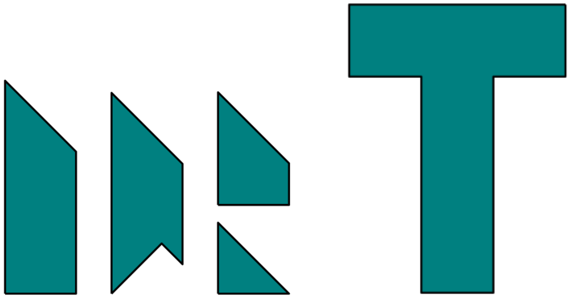
Each package has its own way of reporting the solutions. CPLEX output uses the XML format. If we use CPLEX to solve the Reid problem, then each of the 4 solutions will be stored, along with much more information.

Here is what one Reid solution looks like:

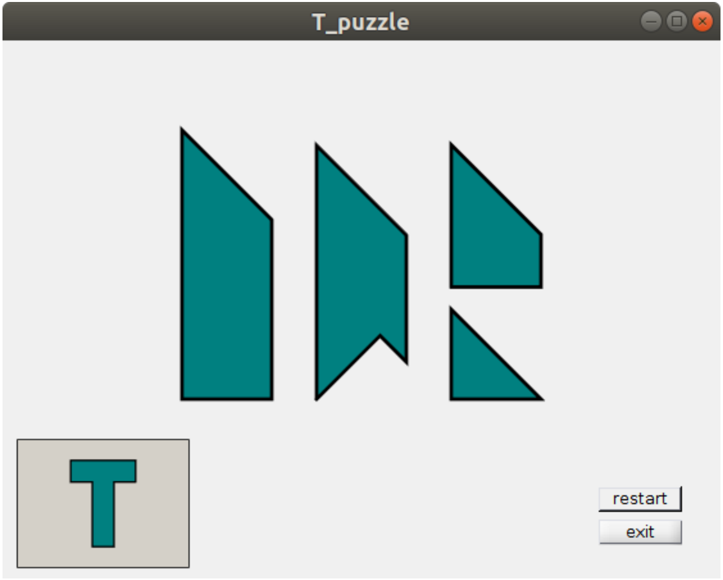
```
<CPLEXSolution>
   ...
<variables>
 <variable name="x2" index="4" value="1"/>
 <variable name="x6" index="14" value="1"/>
 <variable name="x4" index="8" value="1"/>
 <variable name="x5" index="12" value="1"/>
</variables>
</CPLEXSolution>
```

This is the first solution we identified earlier, namely (0,1,0,1,1,1,0,0,0,0).

The T Puzzle



Cleve Moler's T Puzzle GUI



Get a Copy of Cleve Moler's T Puzzle GUI

```
https://people.sc.fsu.edu/~jburkardt/m_src/...
  t_puzzle_gui/t_puzzle_gui.m
```

1. Challenge (T, Arrow, Rhombus, Fat Arrow)
2. Select a Piece
3. Rotate a Piece
4. Flip a Piece
5. Drag a Piece
6. Solve...?

The Humpty Dumpty Problem

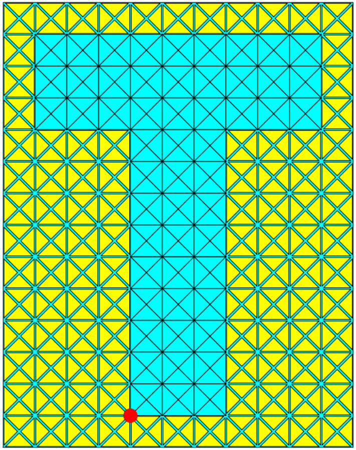
If the T puzzle is similar to the Reid problem, then presumably we can construct a procedure that will automatically find all possible solutions.

But:

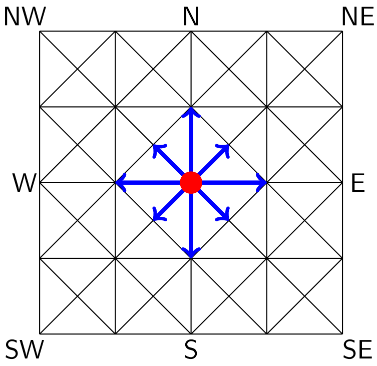
- Each tile is a different shape;
- The tiles are not simple rectangular shapes;
- Some tiles have more reflections and rotations than others;
- A rectangular grid of cells won't work; we have diagonal lines too.
- The resulting linear system will be much larger than for Reid.

T Puzzle Grid

To accommodate various rotations, reflections, translations of the tiles, we need a grid of $6 \times 9 \times 4$ isosceles right triangles: 6 big squares, each containing 9 little squares, each containing 4 triangles.

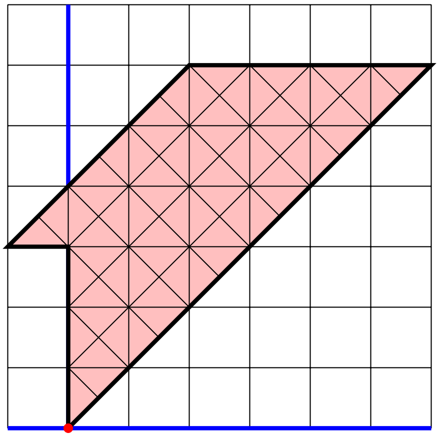


Boundary Word Compass



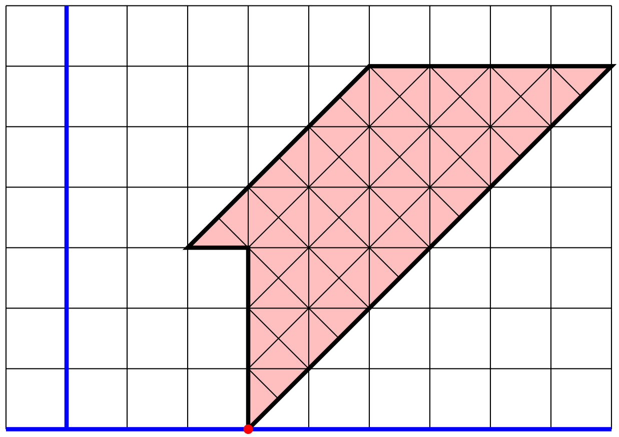
From red dot, there are 8 possible directions.
N, S, E, W steps of length 1.
NE, NW, SE, SW steps of length $\frac{\sqrt{2}}{2}$

Boundary Word Description



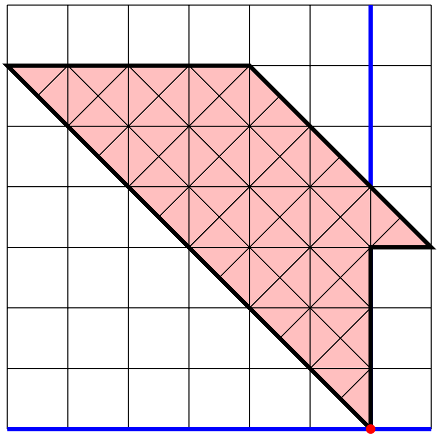
```
Start at P=(0,0) (red dot)
Boundary path = 12 NE + 4 W + 6 SW + 1 E + 3 S
```

Translate Boundary Word



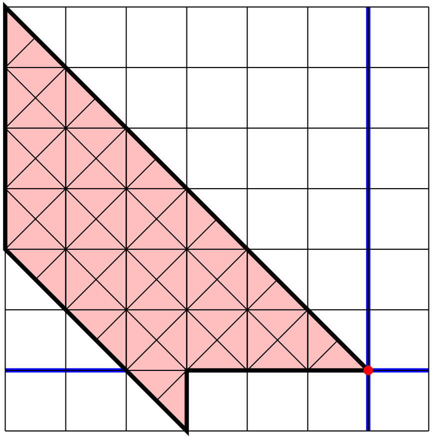
```
Translate by (3,0): Start at P+(3,0) (red dot)
Boundary path = 12 NE + 4 W + 6 SW + 1 E + 3 S
```

Reflect Boundary Word (Swap West and East)



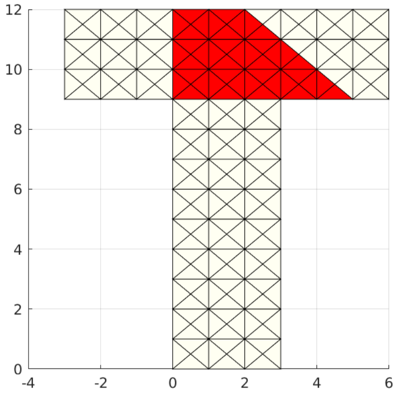
```
Reflect: Start at P (red dot)
Old Boundary path: 12 NE + 4 W + 6 SW + 1 E + 3 S
New Boundary path: 12 NW + 4 E + 6 SE + 1 W + 3 S
```

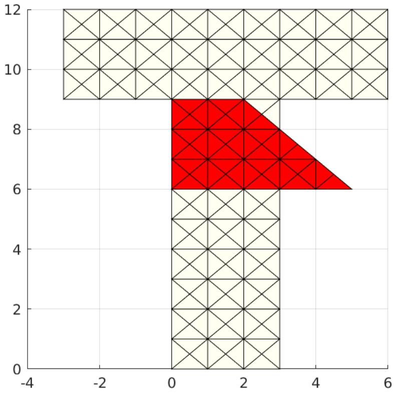
Rotate (90^o : $N \to W, W \to S, S \to E, E \to N$)



```
Rotate 90: Start at P (red dot)
Old Boundary path: 12 NE + 4 W + 6 SW + 1 E + 3 S
New Boundary path: 12 NW + 4 S + 6 SE + 1 N + 3 E
```

The Whole Tile Must Be Inside!



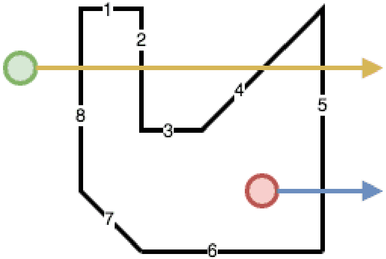


Tile in Grid → Element Centroids in Grid

Because the grid and the tiles may be of strange, irregular, nonconvex shape, we need to check that every triangular element of a tile is inside the grid.

It suffices to verify that the centroid of each element is inside the grid.

A formula known as *point in polygon* can give us the answer. The surrounding polygon does not need to be convex, but it can't be "weird" (as in, the boundary crossing over itself).



Linear System for T Puzzle

Now we have the tools we need to build the linear system $A * x = b$.

There are $6 \times 9 \times 4 = 216$ triangular elements. Each of them must be covered exactly once. There are 4 tiles, each must be used exactly once. This gives a total of 216+4=220 equations,

Each tile configuration (rotation + reflection + translation remaining in grid) is a variable. The tiles vary in their number of configurations:

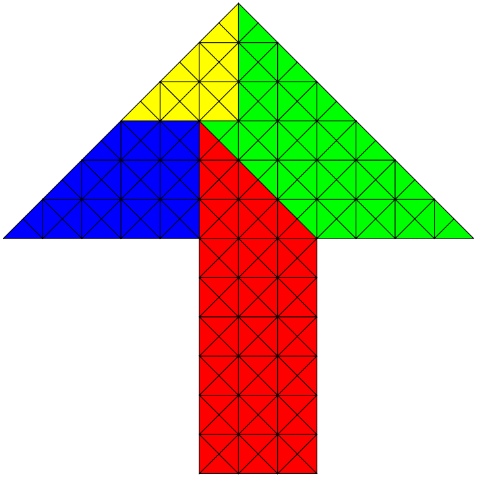
1	20	long
2	2	weird
3	56	trapezoid
4	70	triangle
Total	148	The T

So we have to solve a 220×148 linear system, whose answers x can only have the values 0 and 1.

We write the linear system as an LP file. In a few seconds, Gurobi returns 2 possible solutions. One is simply the left/right reflection of the other.

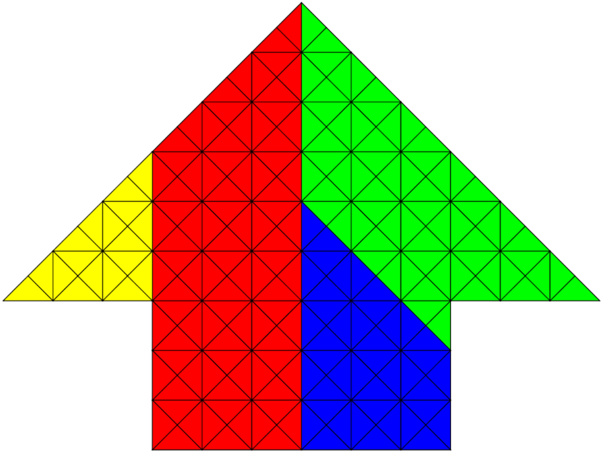
Arrow solution

The Arrow gurobi solution



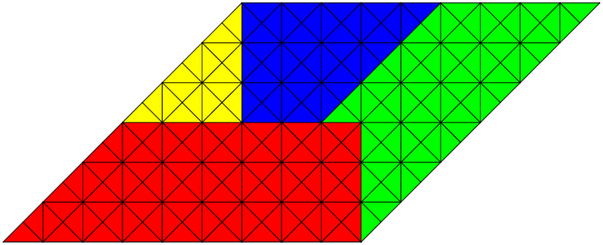
Fat Arrow solution

The Fat T gurobi solution



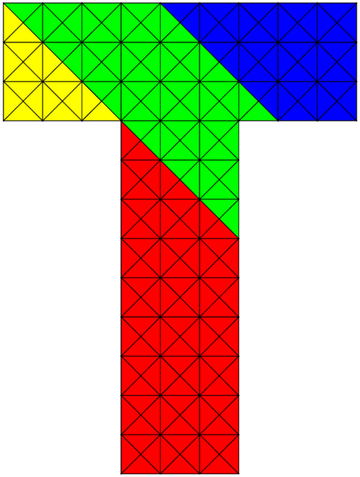
Rhombus solution

The Rhombus gurobi solution

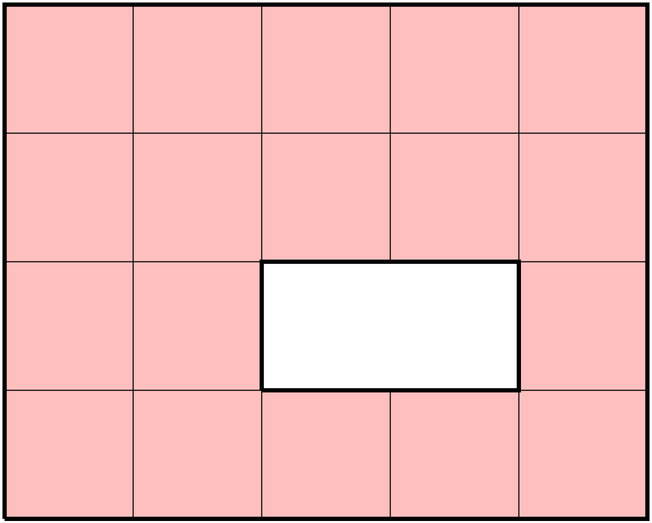


T solution

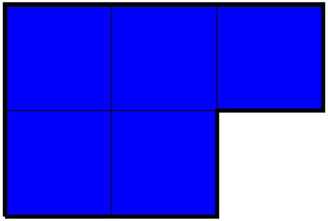
The T gurobi solution

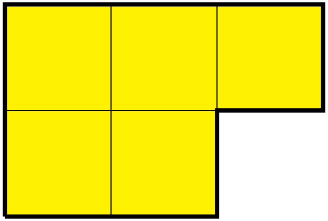


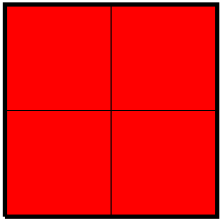
4x5 Example: Region R

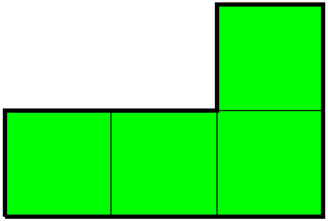


4x5 Example: Tiles





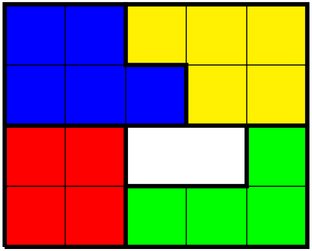


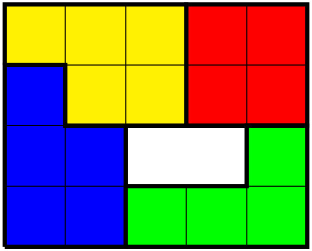


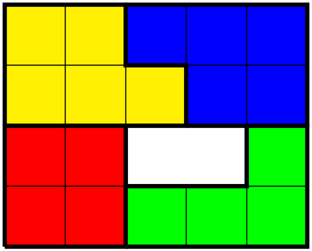
4x5 Example: Run the code

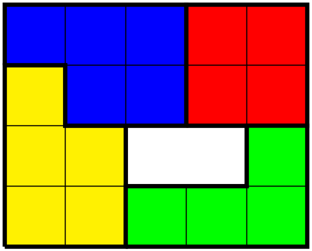
- RREF system has 23 rows and 62 columns;
- Augmented system has 42 degrees of freedom;
- ALL binary right hand sides is 2^{42}, on the order of a **trillion**;
- Only check binary RHS with at most 4 degrees of freedom set to 1: $1 + 42 + 42 * 41/2 + 42 * 41 * 40/6 + 42 * 41 * 40 * 39/24 = 124,314$;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

4x5 Example: Four Solutions

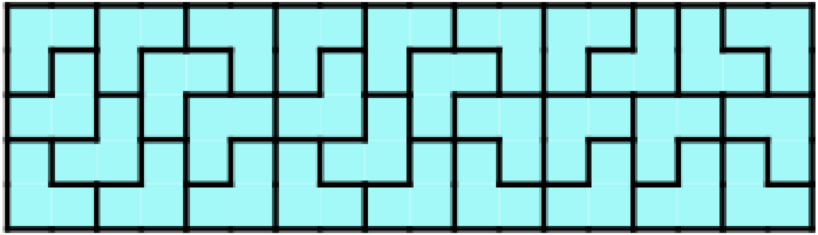








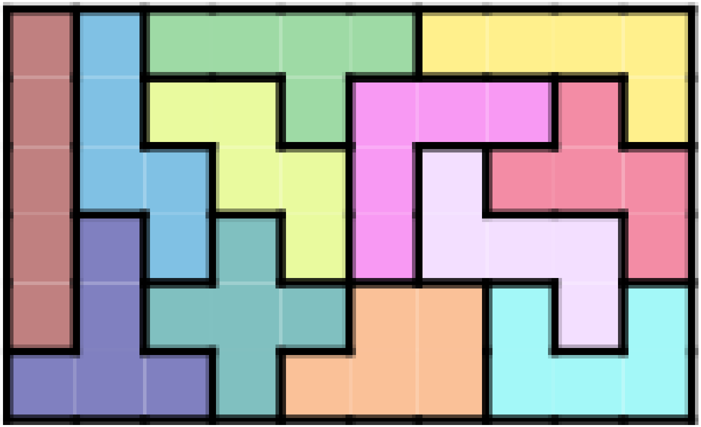
Tiling by Multiple Copies of a Single Polyomino



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

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

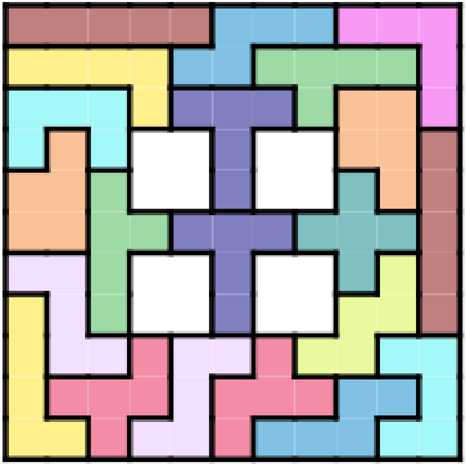
Tiling by One Copy of Each Pentomino



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

9,356 solutions computed by CPLEX in 7.3 minutes.

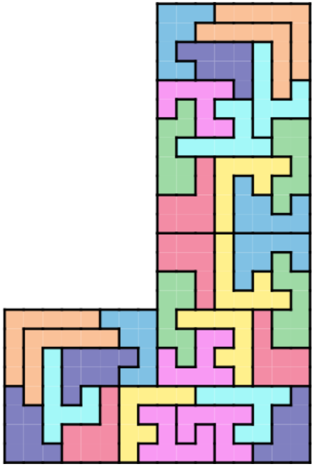
Tiling a Region with Holes



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

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

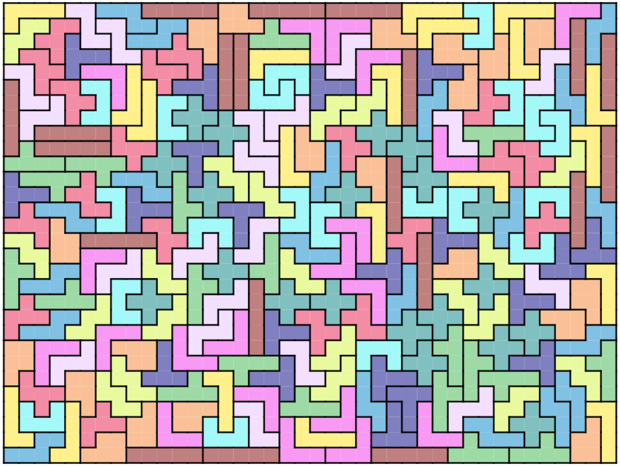
Tiling a Nonrectangular Region



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

1 solution computed by CPLEX in 13 minutes.

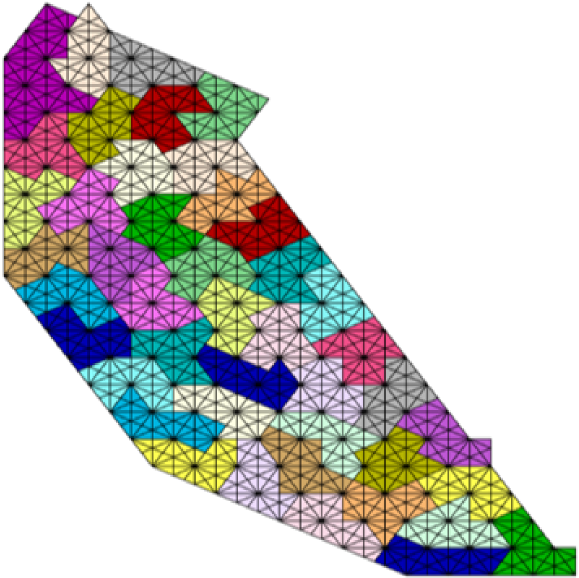
A Large Problem



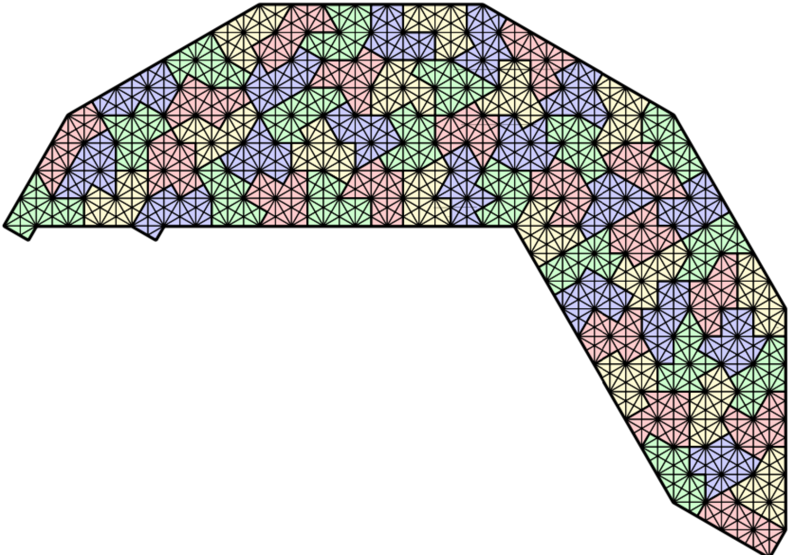
12 pentominoes, 20 copies each, 1,213 equations, 67,396 variables

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

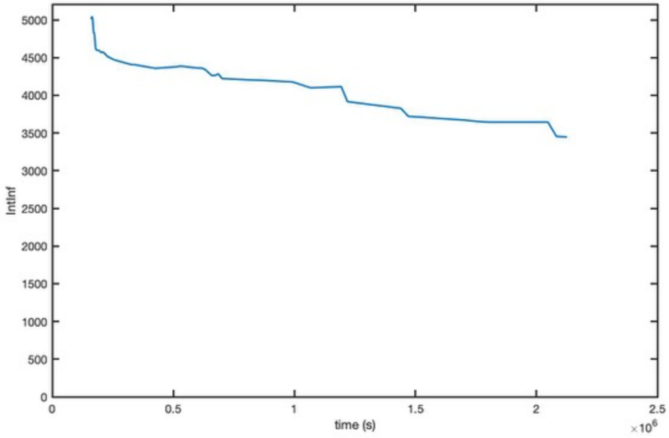
Tortoise: A piece of Eternity (45/209 tiles)



Boomerang: A piece of Eternity (66/209 tiles)



Trying the whole Eternity puzzle (209/209 tiles)



Trying the whole Eternity puzzle (209/209 tiles)

We are running the Eternity problem on Gurobi.
Gurobi reduced our LP problem to a kernel of about 5,000 constraints.
While it works on them, it can display a plot of its progress.

1500 constraints done in 2,000,000 seconds
rate = 1333 seconds per constraint
5000 constraints * 1333 seconds = 6,665,000 seconds total.
6,665,000 seconds / 86400 seconds per day = 77 days.

Optimism: *Could get a positive result in 6 more weeks.*

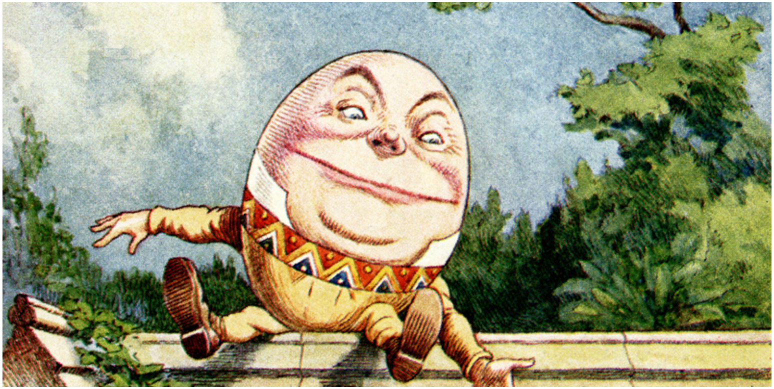
Conclusion: Rebuilding with Linear Algebra

To solve a tiling problem, we look for an underlying grid of cells that define both the region and the tiles. *This isn't always possible!*

- Equations: Each region cell must be covered, just once.
- Equations: Each tile must be used, just once.
- Variables: Each rotated, reflected, translated tile remaining in region
- Equations + Variables: **underdetermined linear system** $Ax = b$.
- **Reduced Row Echelon Form** lets us analyze the system.
- **Linear Programming Software** solves big systems.
- We seek **binary** vectors x whose entries are only 0 or 1.
- There may be no such solutions at all.
- If there are **free variables**, we may have multiple solutions.

Any solution x tells us exactly how to use the pieces so we can put a broken object back together...

The Humpty Dumpty Solution



Humpty Dumpty thought he was through,
But linear algebra knew what to do.
It set up the system and solved it and then
Neatly put Humpty together again.

References

Cleve Moler's T Puzzle:

https://people.sc.fsu.edu/∼jburkardt/m_src/t_puzzle_gui/t_puzzle_gui.m

https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/moler/exm/chapters/puzzle.pdf

Online references:

https://people.sc.fsu.edu/∼jburkardt/presentations/t_puzzle_2022_pitt.pdf

https://people.sc.fsu.edu/∼jburkardt/m_src/t_puzzle/t_puzzle.html

Journal article:

Marcus Garvie, John Burkardt,
A New Mathematical Model for Tiling Finite Regions of the Plane with Polyominoes,
Contributions to Discrete Mathematics,
Volume 15, Number 2, July 2020, pages 95-131.