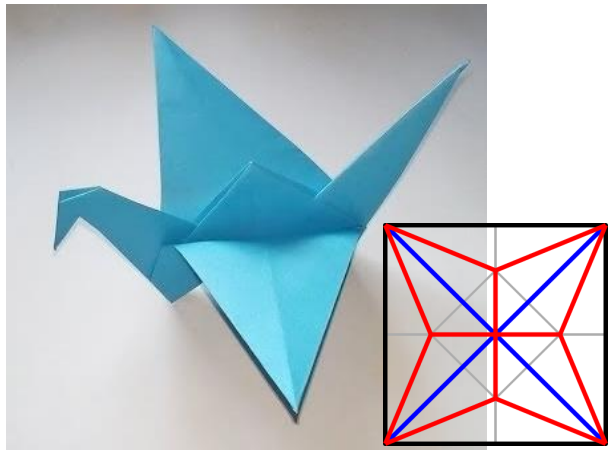
Is this the future of 22.5 design?

Brandon Wong

Ann Arbor, Michigan, USA CFC 6 May 14-17, 2026

Contents

22.5 is everywhere



Traditional

Shuki Kato

Winston Lee

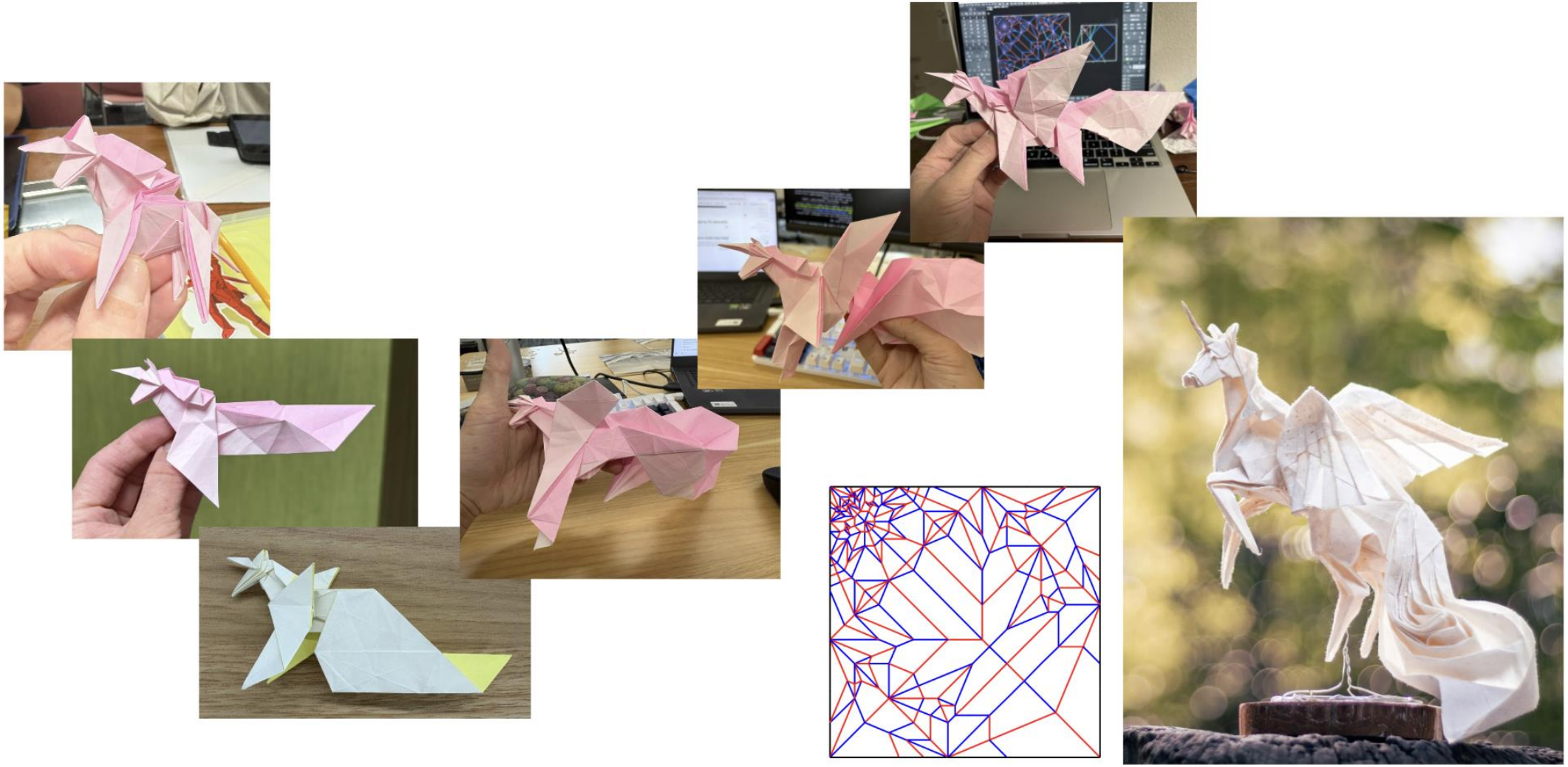
John Montroll

Beth Johnson

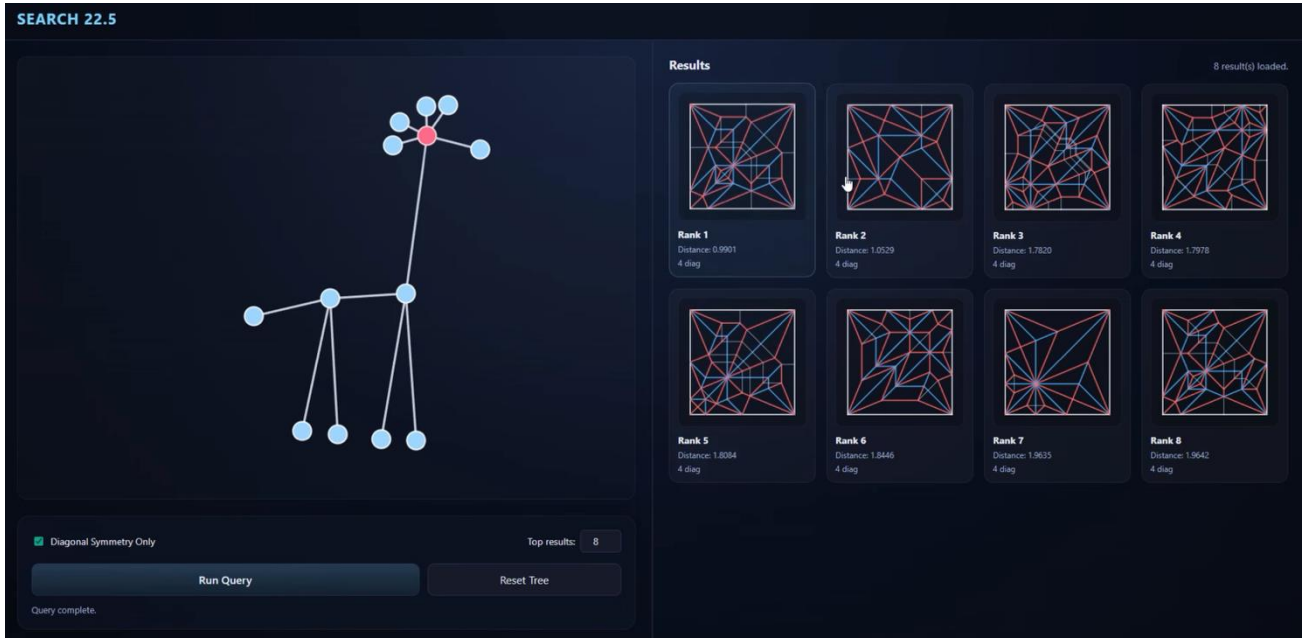
22.5 is hard (but worth it)

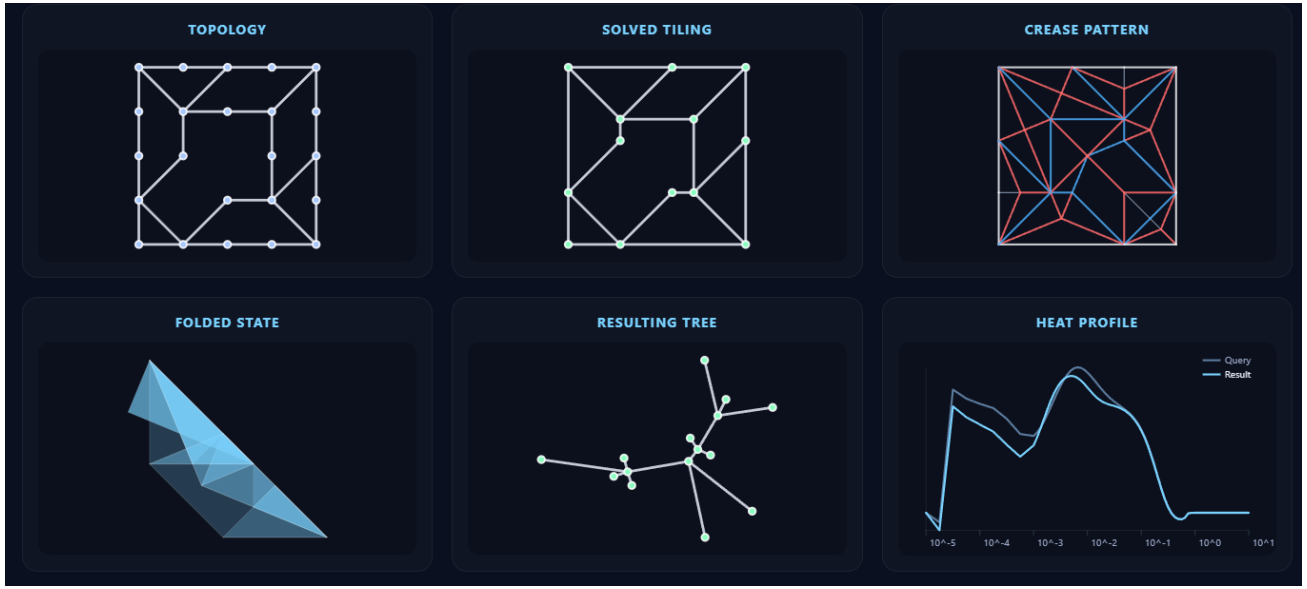
- Not fully integer, not fully continuous $a + b\sqrt{2}$
- Generally considered the most organic and the most enjoyable to fold
- **No well-defined forward design methods have been found**

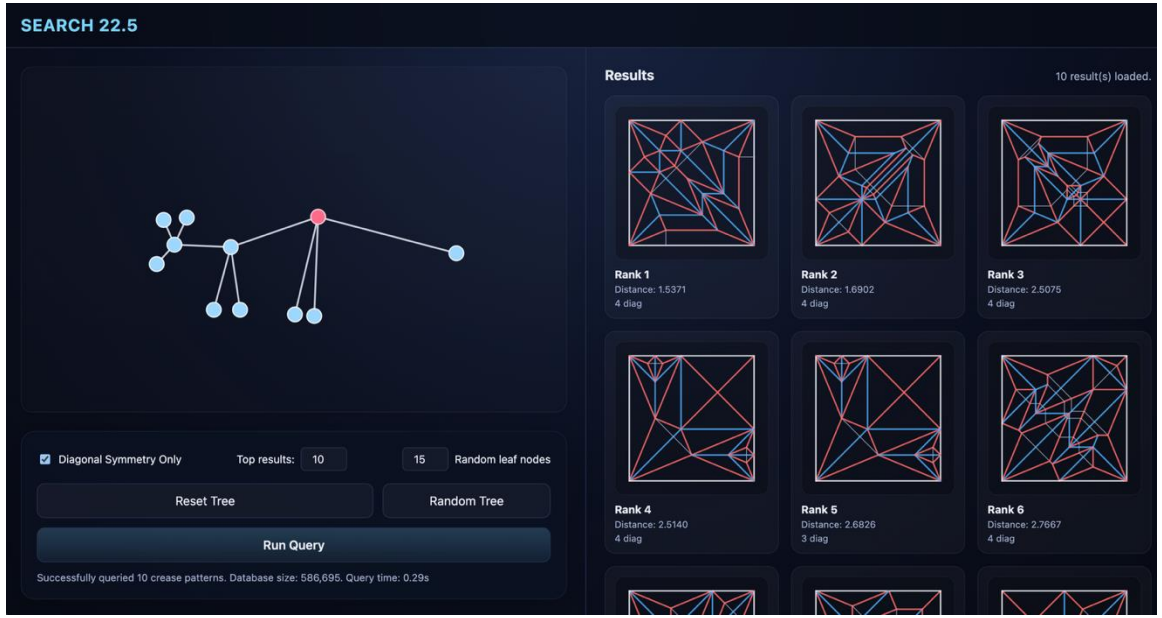
Typical 22.5 design process

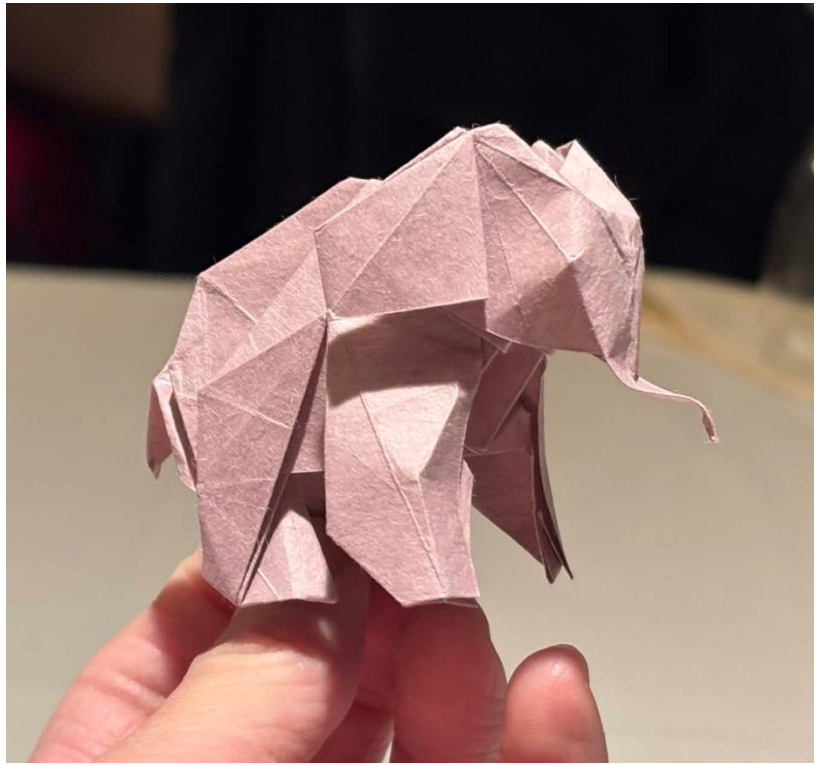


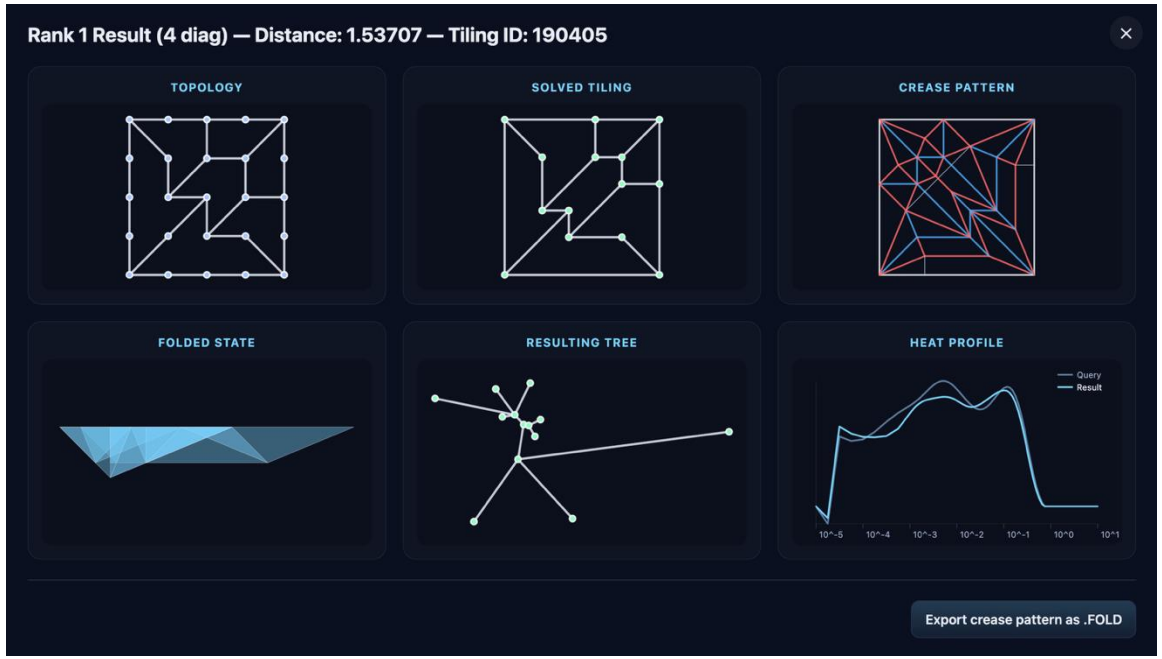
Demo

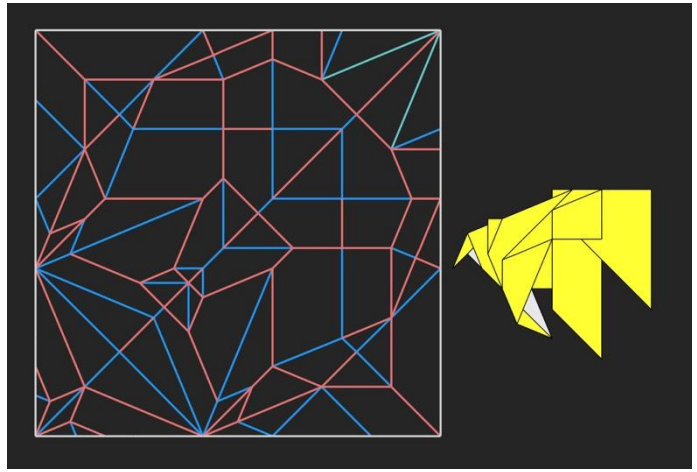

SEARCH 22.5
Results
8 result(s) loaded.
Rank 1
Distance: 0.9901
4 diag
Rank 2
Distance: 1.0529
4 diag
Rank 3
Distance: 1.7820
4 diag
Rank 4
Distance: 1.7978
4 diag
Rank 5
Distance: 1.8084
4 diag
Rank 6
Distance: 1.8446
4 diag
Rank 7
Distance: 1.9635
4 diag
Rank 8
Distance: 1.9642
4 diag
Diagonal Symmetry Only
Top results: 8
Run Query
Reset Tree
Query complete.

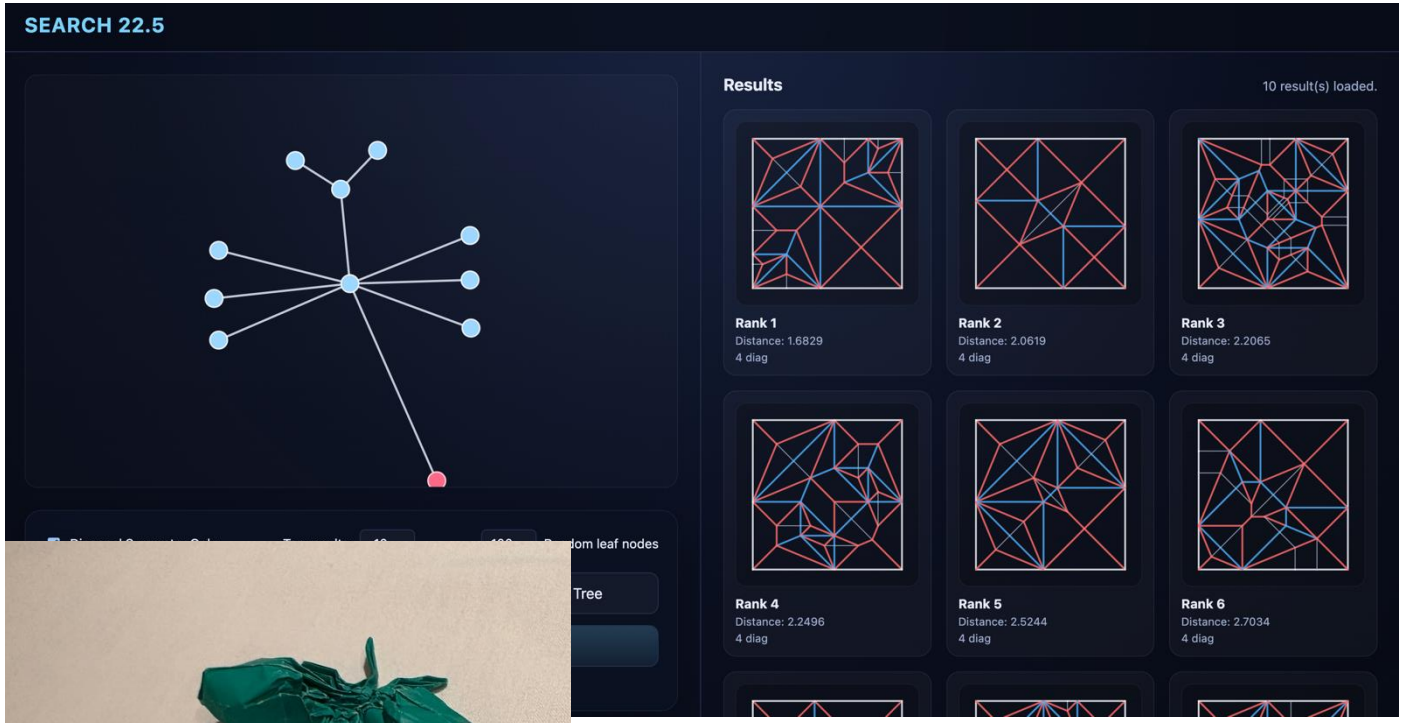

TOPOLOGY
SOLVED TILING
CREASE PATTERN
FOLDED STATE
RESULTING TREE
HEAT PROFILE
Query
Result
10^-5
10^-4
10^-3
10^-2
10^-1
10^0
10^1

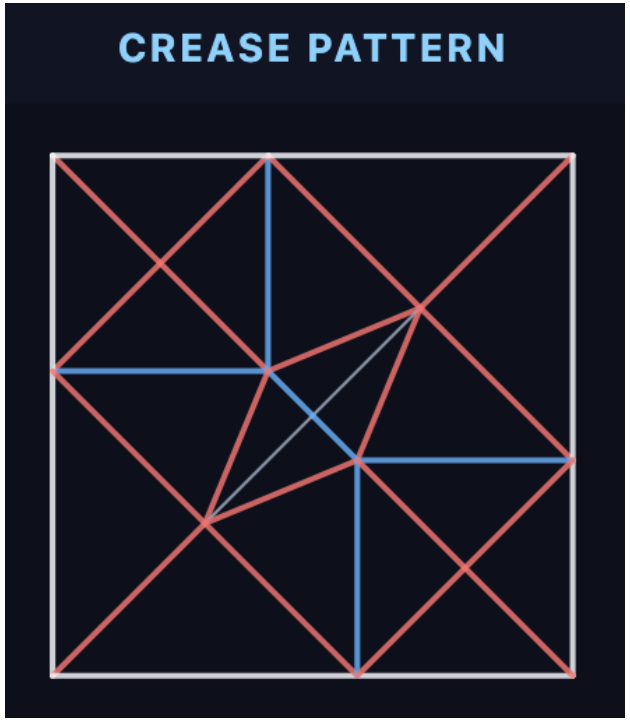

SEARCH 22.5
Results
10 result(s) loaded.
Rank 1
Distance: 1.5371
4 diag
Rank 2
Distance: 1.6902
4 diag
Rank 3
Distance: 2.5075
4 diag
Rank 4
Distance: 2.5140
4 diag
Rank 5
Distance: 2.6826
3 diag
Rank 6
Distance: 2.7667
4 diag
Diagonal Symmetry Only
Top results:
10
15
Random leaf nodes
Reset Tree
Random Tree
Run Query
Successfully queried 10 crease patterns. Database size: 586,695. Query time: 0.29s

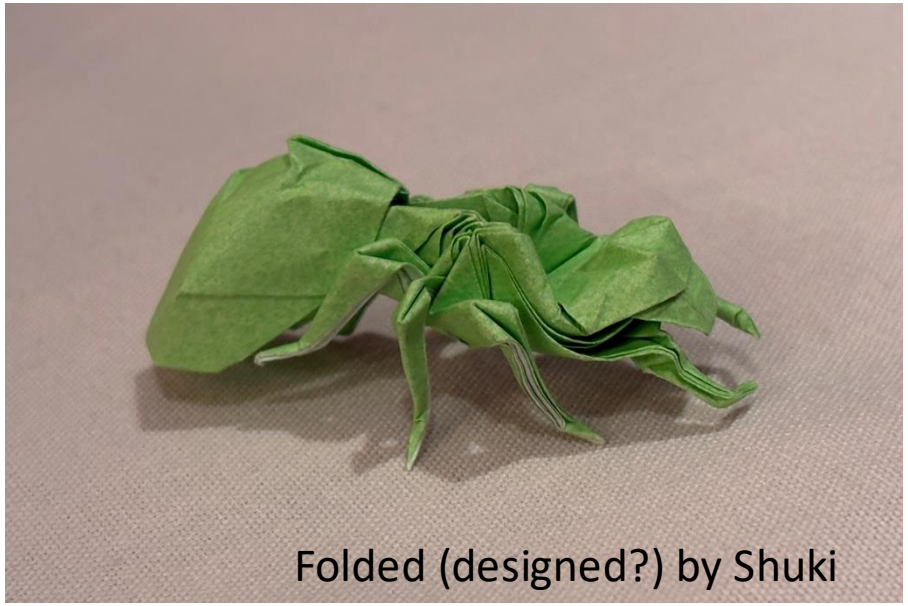


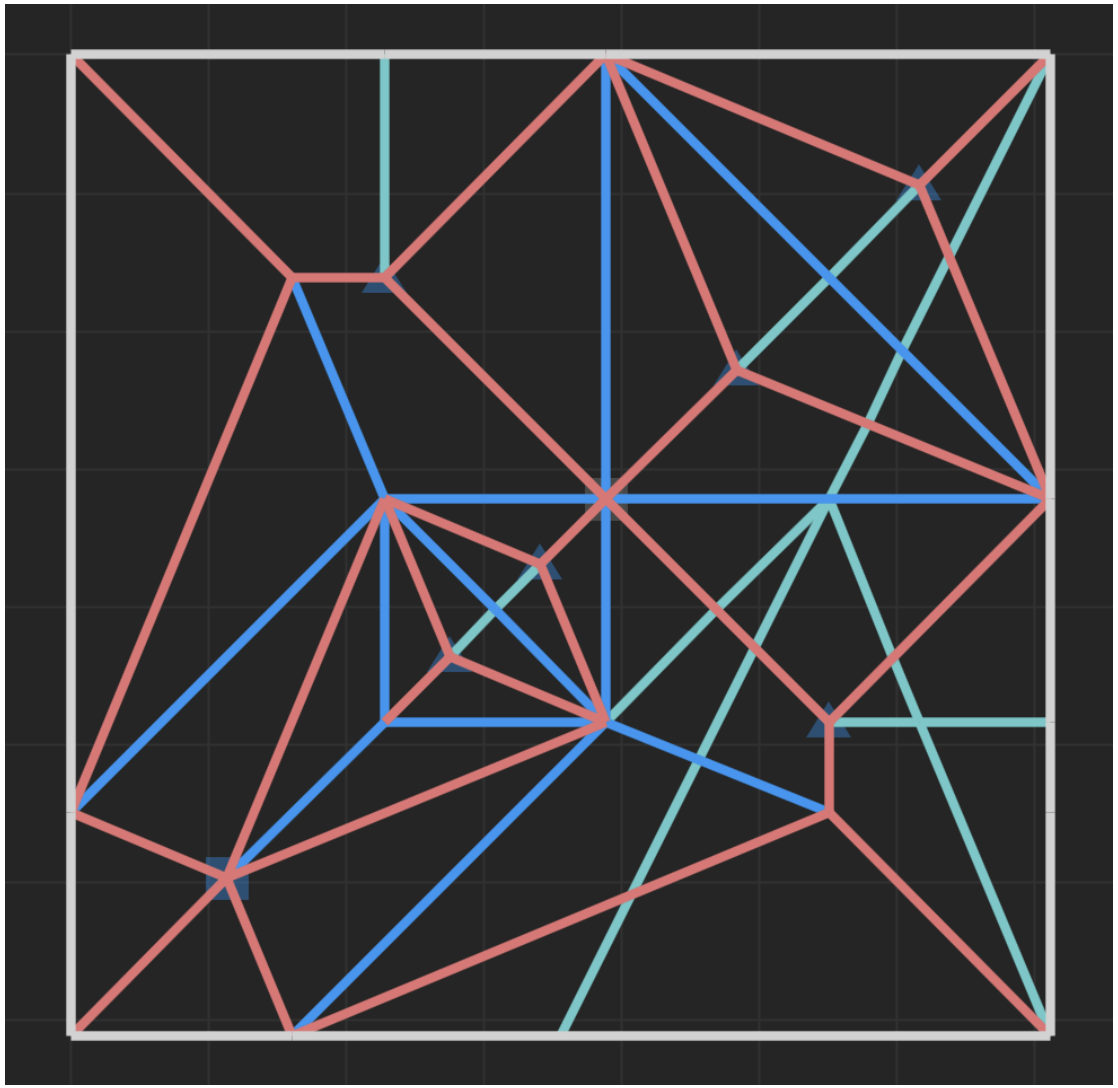

Rank 1 Result (4 diag) — Distance: 1.53707 — Tiling ID: 190405
TOPOLOGY
SOLVED TILING
CREASE PATTERN
FOLDED STATE
RESULTING TREE
HEAT PROFILE
Query
Result
10^-5
10^-4
10^-3
10^-2
10^-1
10^0
10^1
Export crease pattern as .FOLD



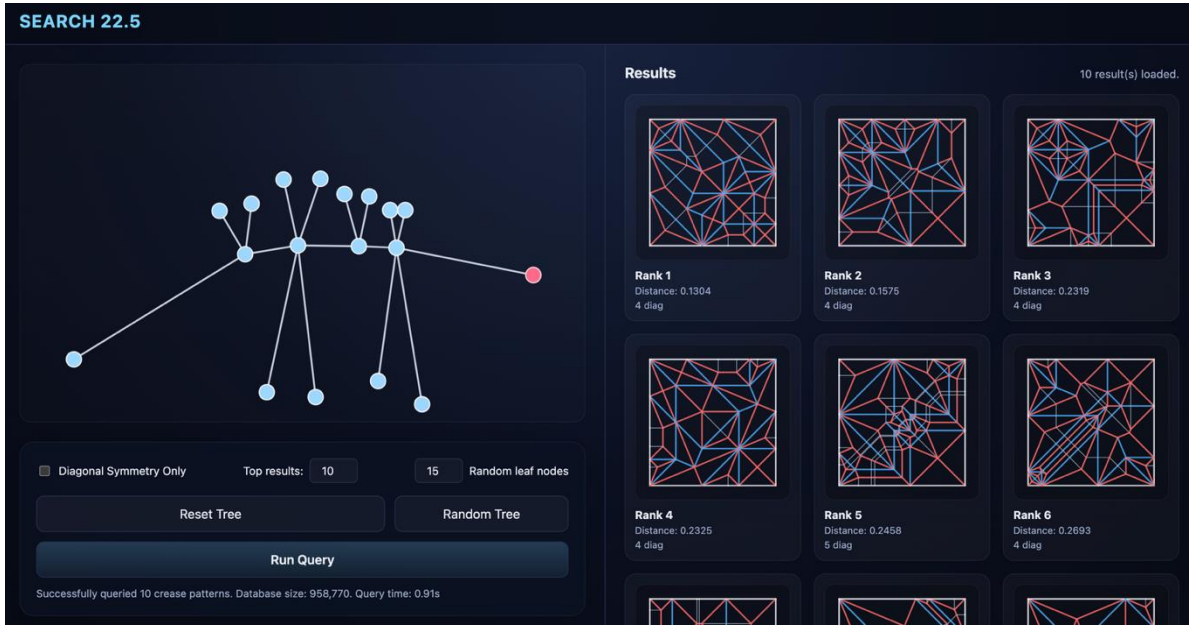

SEARCH 22.5
Results
10 result(s) loaded.
Rank 1
Distance: 1.6829
4 diag
Rank 2
Distance: 2.0619
4 diag
Rank 3
Distance: 2.2065
4 diag
Rank 4
Distance: 2.2496
4 diag
Rank 5
Distance: 2.5244
4 diag
Rank 6
Distance: 2.7034
4 diag
Tree

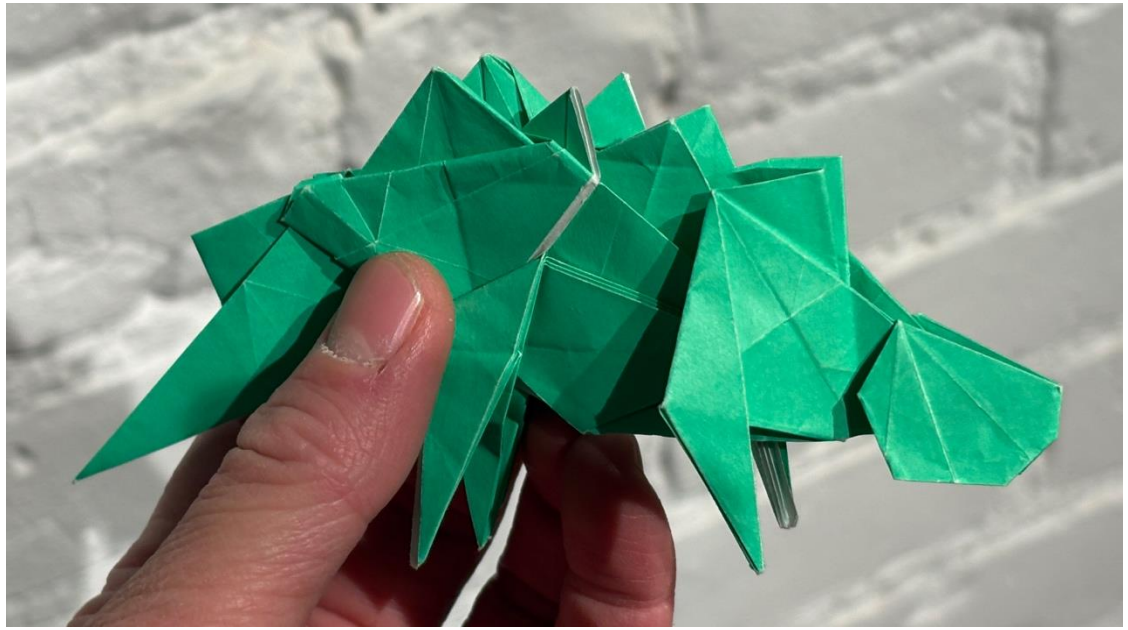

CREASE PATTERN

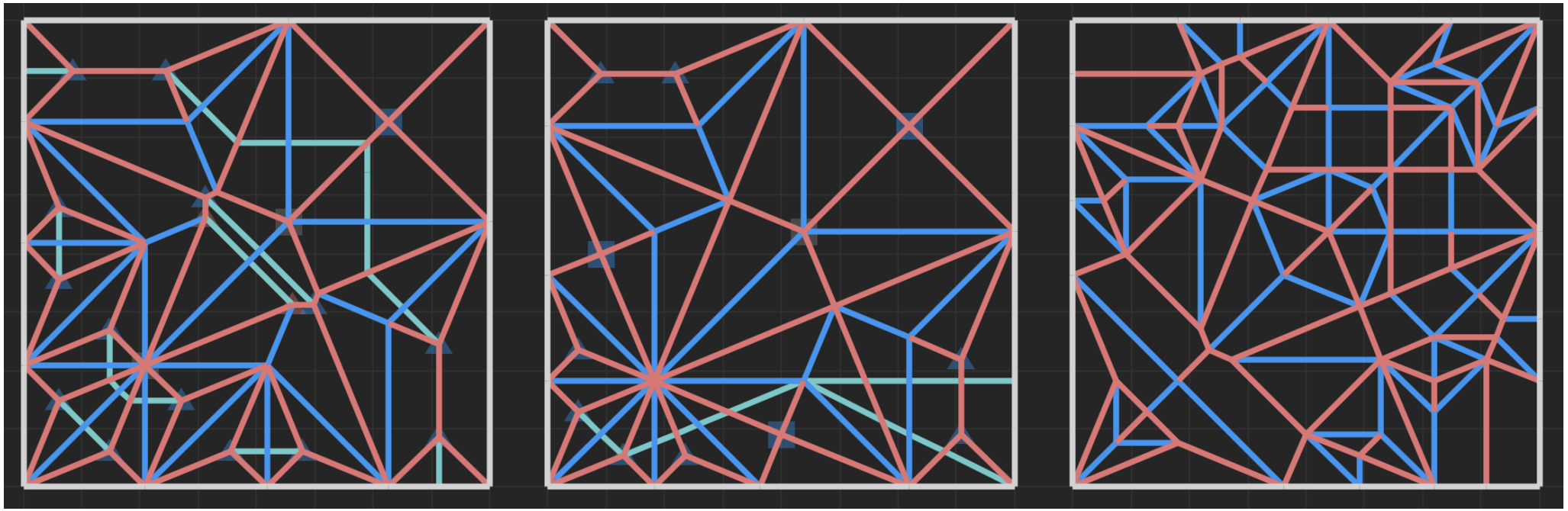

Folded (designed?) by Shuki



Folded (designed?) by Ronik

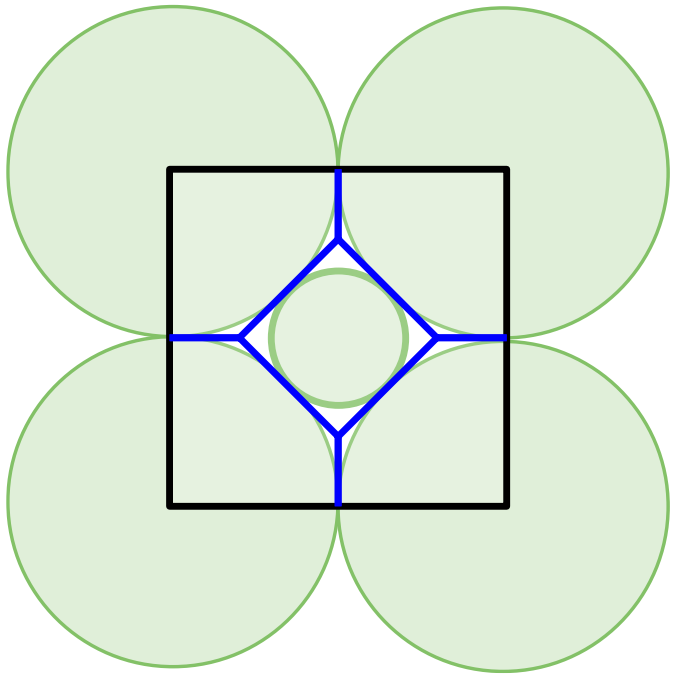

SEARCH 22.5
Results
10 result(s) loaded.
Rank 1
Distance: 0.1304
4 diag
Rank 2
Distance: 0.1575
4 diag
Rank 3
Distance: 0.2319
4 diag
Rank 4
Distance: 0.2325
4 diag
Rank 5
Distance: 0.2458
5 diag
Rank 6
Distance: 0.2693
4 diag
Diagonal Symmetry Only
Top results:
10
15
Random leaf nodes
Reset Tree
Random Tree
Run Query
Successfully queried 10 crease patterns. Database size: 958,770. Query time: 0.91s



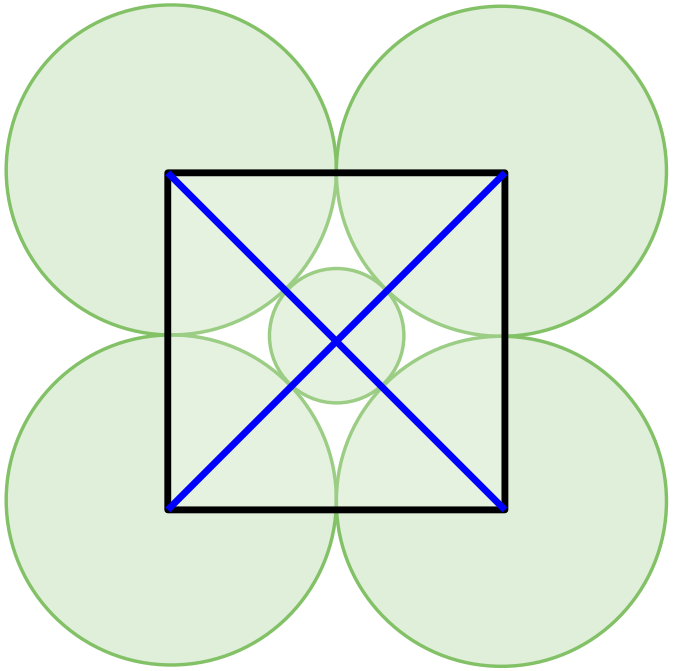


Before I reveal how it works…

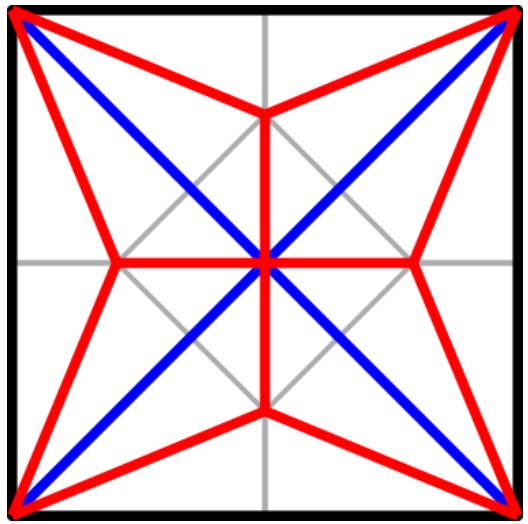
Hint 1: **Packing, or tiling?**



Packing

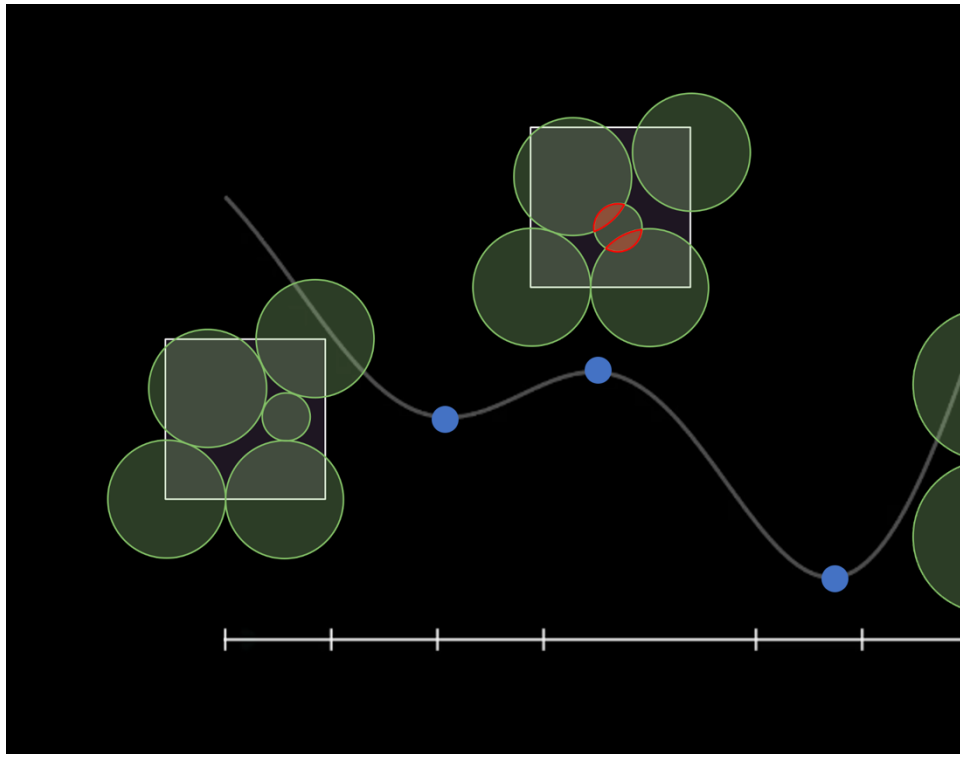


Tiling

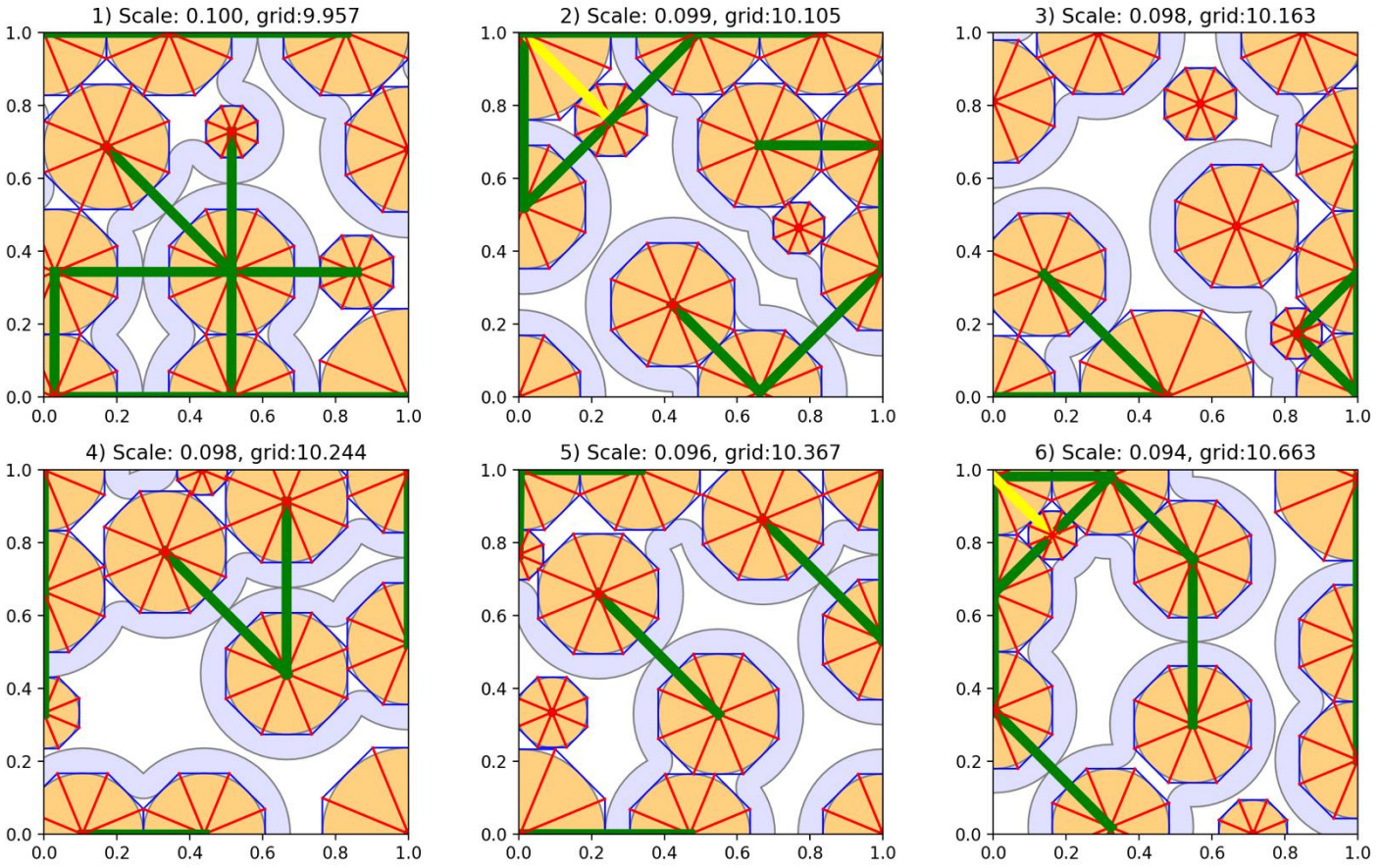


Crease pattern

Hint 1: **Packing, or tiling?**

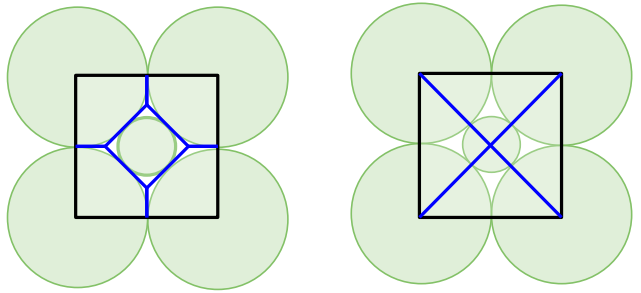


Remember how Treemaker works?

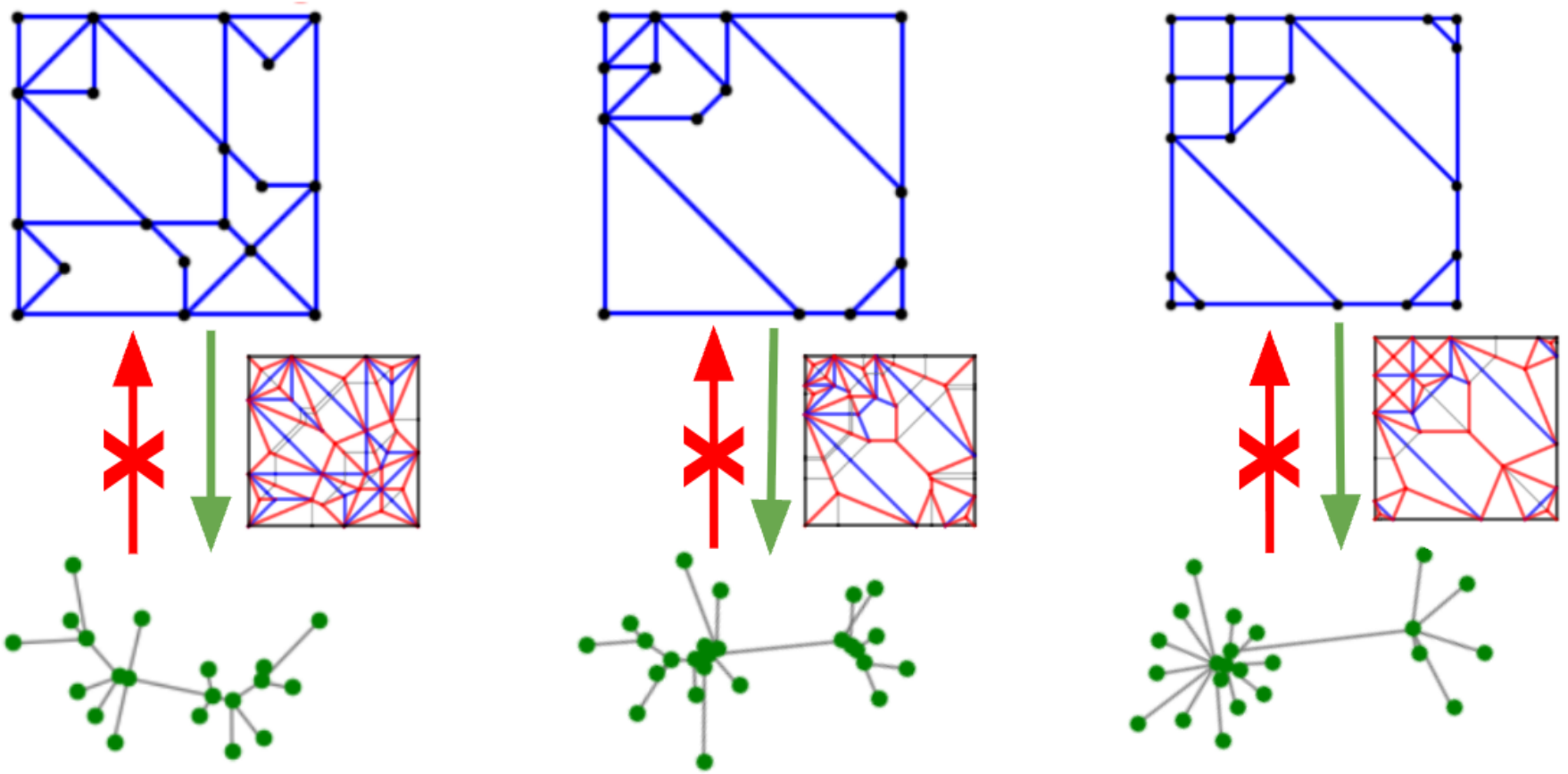


Attempting to modify circle packing
to octagon packing doesn't really work

Hint 1: **Packing, or tiling?**



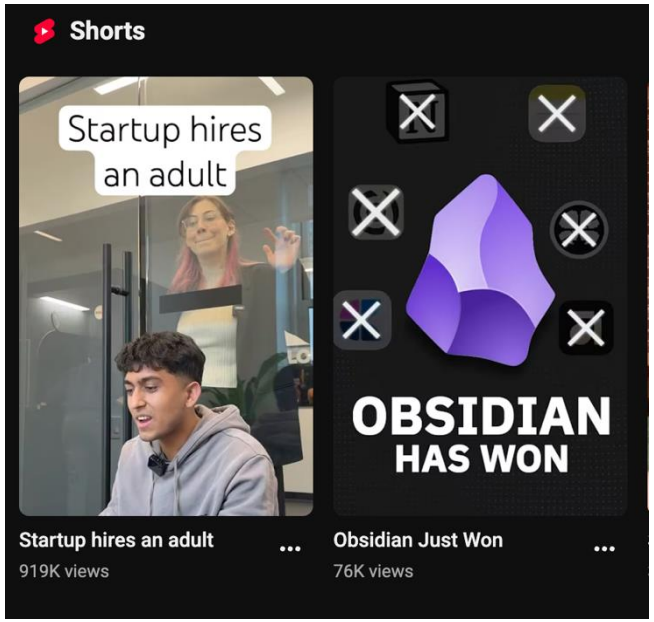
Packing Tiling

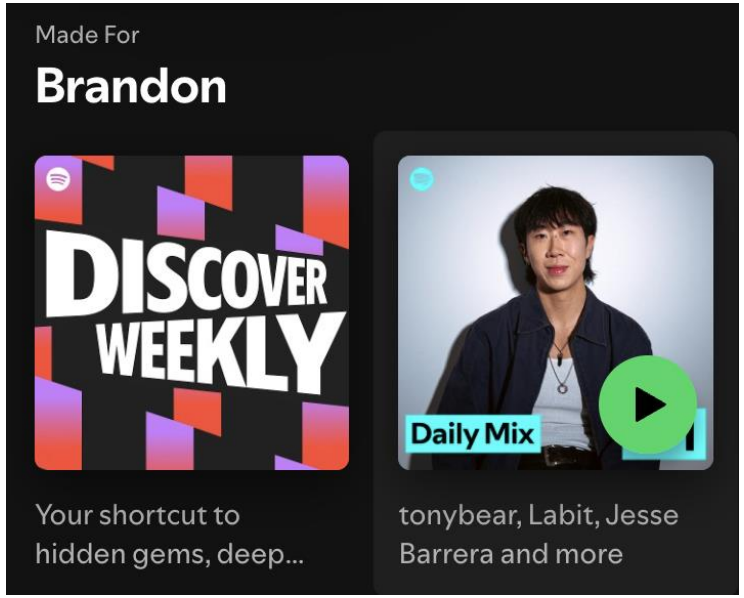


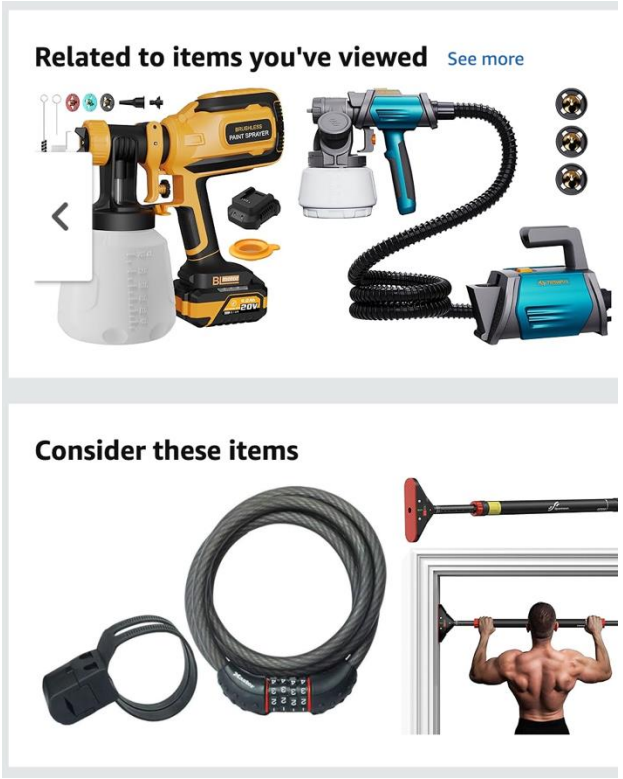
It’s possible to convert a tiling into a crease pattern, and a crease pattern into a tree

But it’s impossible to solve the reverse

Hint 2: **Online recommendations**

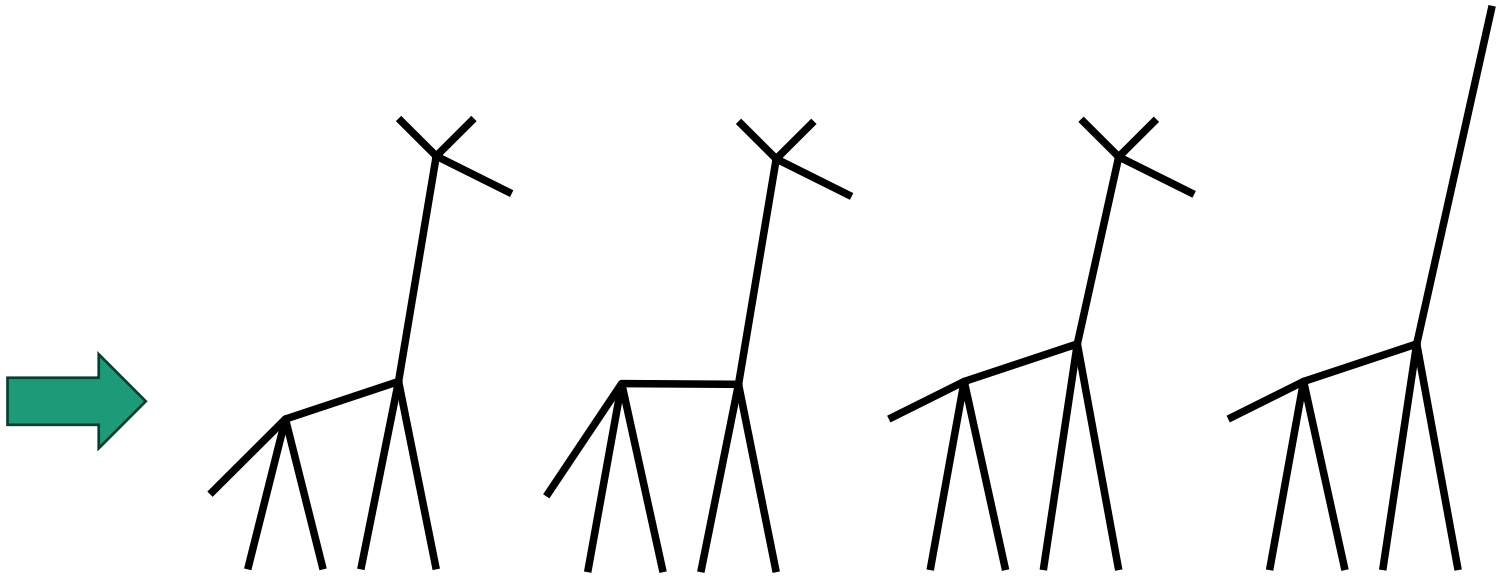






Instantly search through millions of videos/songs/products for the best guess of what you want to see

Hint 3: **Approximations are ok**

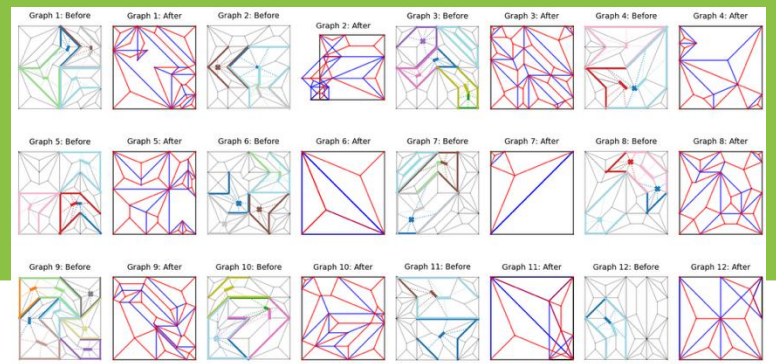


Any of these trees would be acceptable

Solution Overview

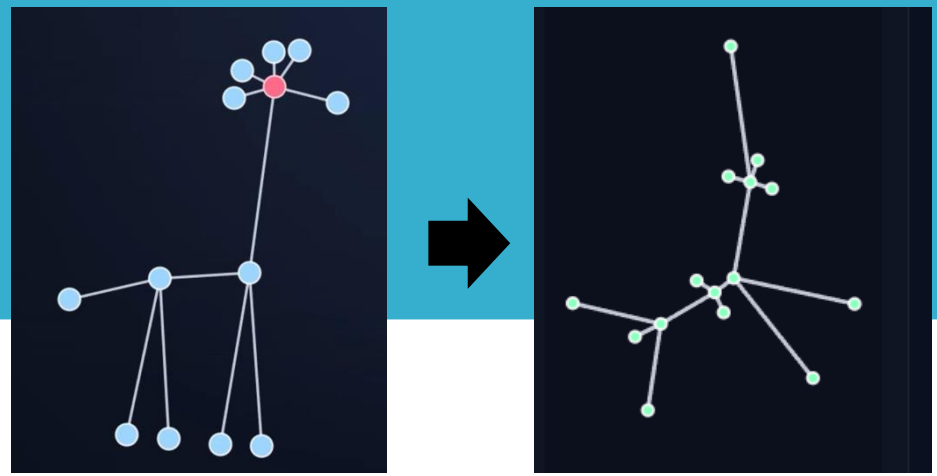
01

Precompute a massive database of crease patterns



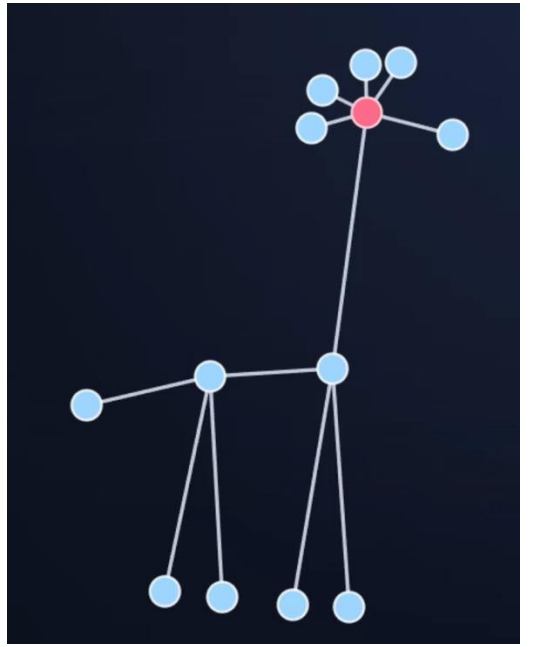
02

Check the database for the closest match



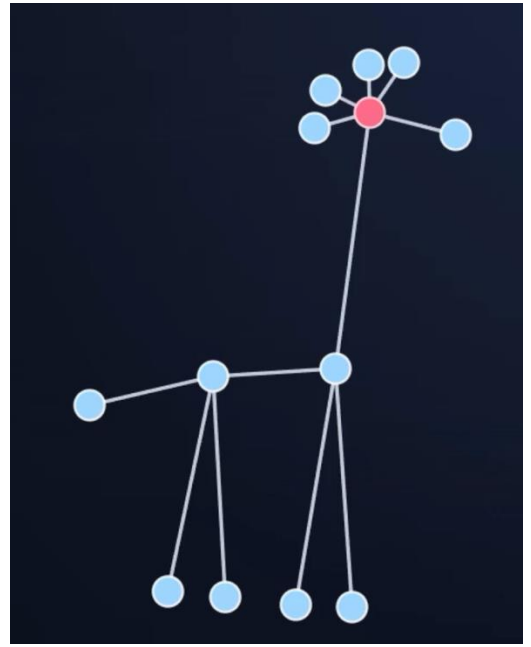
One time: generate a database of crease patterns

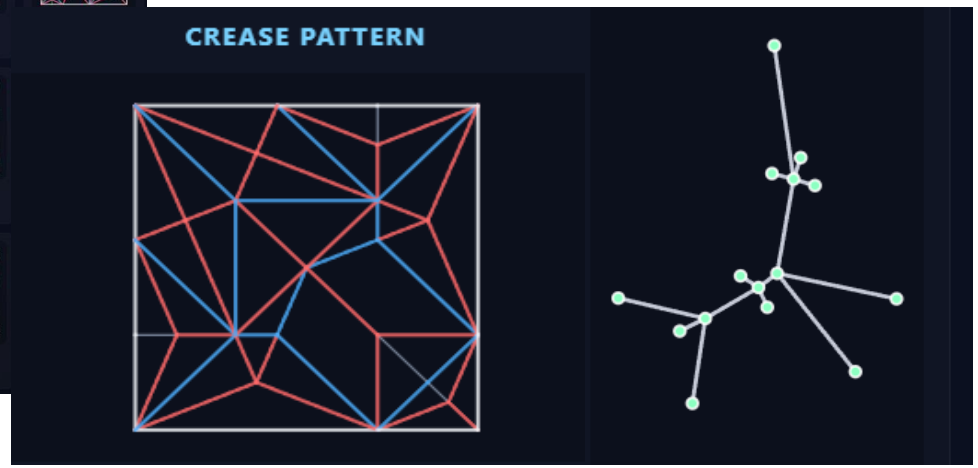
User query tree



One time: generate a database of crease patterns

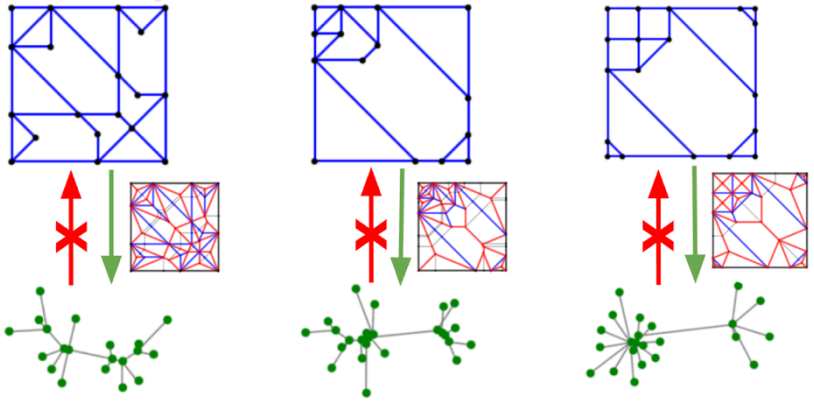
User query tree





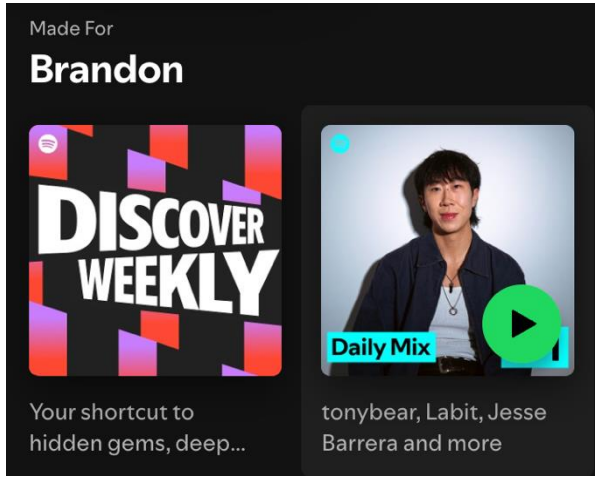
Find the closest match from memory

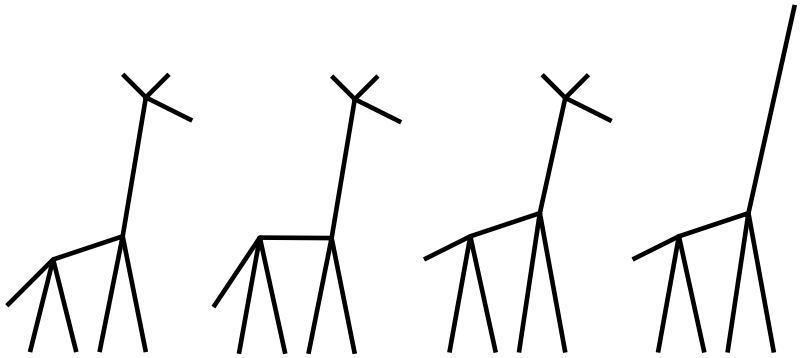
How the hints fit together



We can generate millions of crease patterns and solve their trees, even if we can’t do the reverse

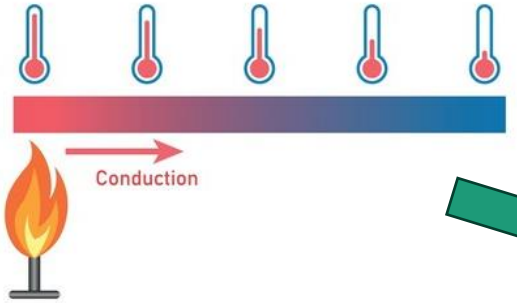
We can use social media algorithms to find crease patterns that might be a good match

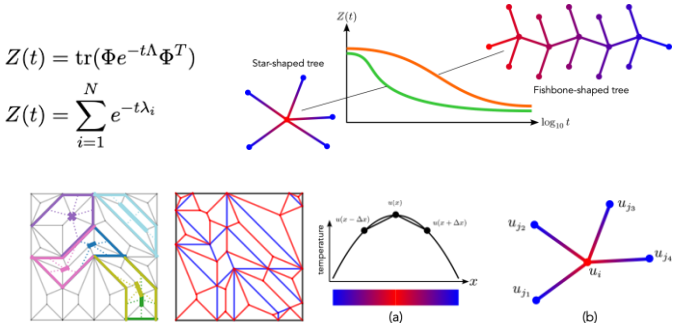


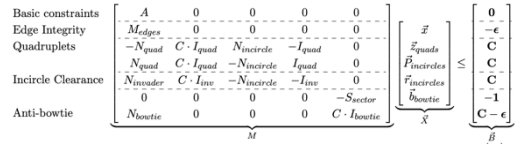


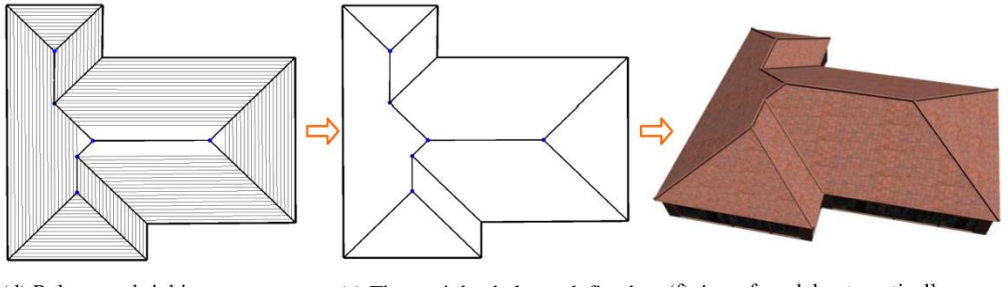
If the match isn’t perfect, it’s ok

Interdisciplinarity









This is a “bilingual” presentation

For the nerds: math details will be in yellow boxes

For everyone else: translations will be in green boxes

Before you fall asleep: main takeaways

- This tool works by pre-generating millions of cps, once. Then, to "design" something, it looks through memory for the closest one

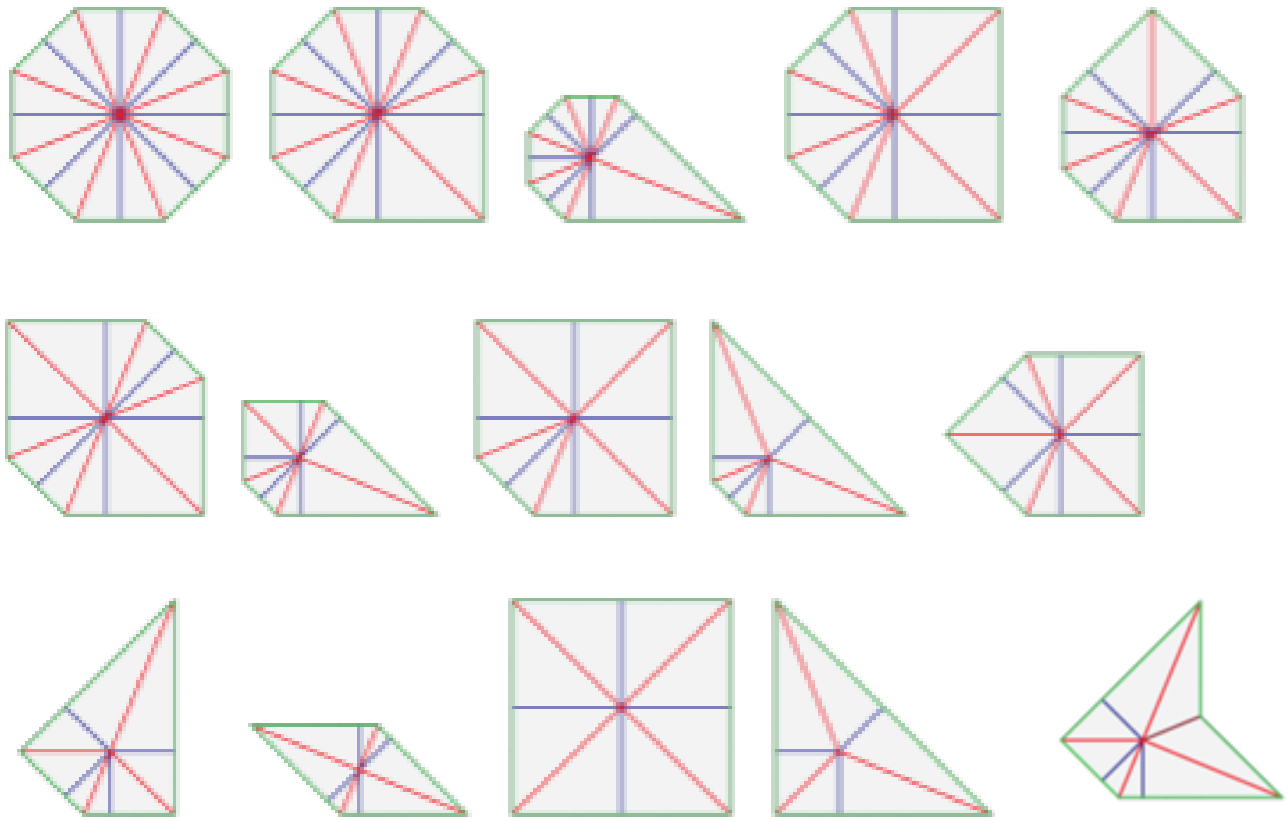
- The math incorporates concepts from many unrelated fields

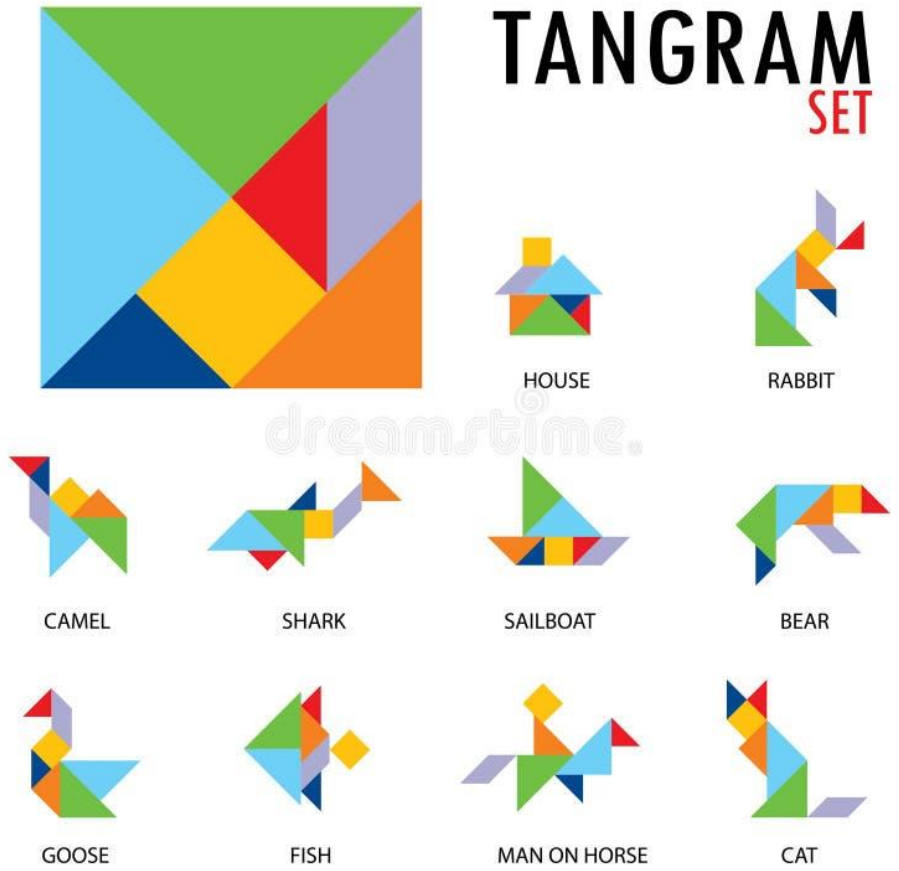
- This will make 22.5 design easier and more accessible, **but without doing all the work for you.**

Database Generation

Goal: generate unlimited supply of valid 22.5 crease patterns

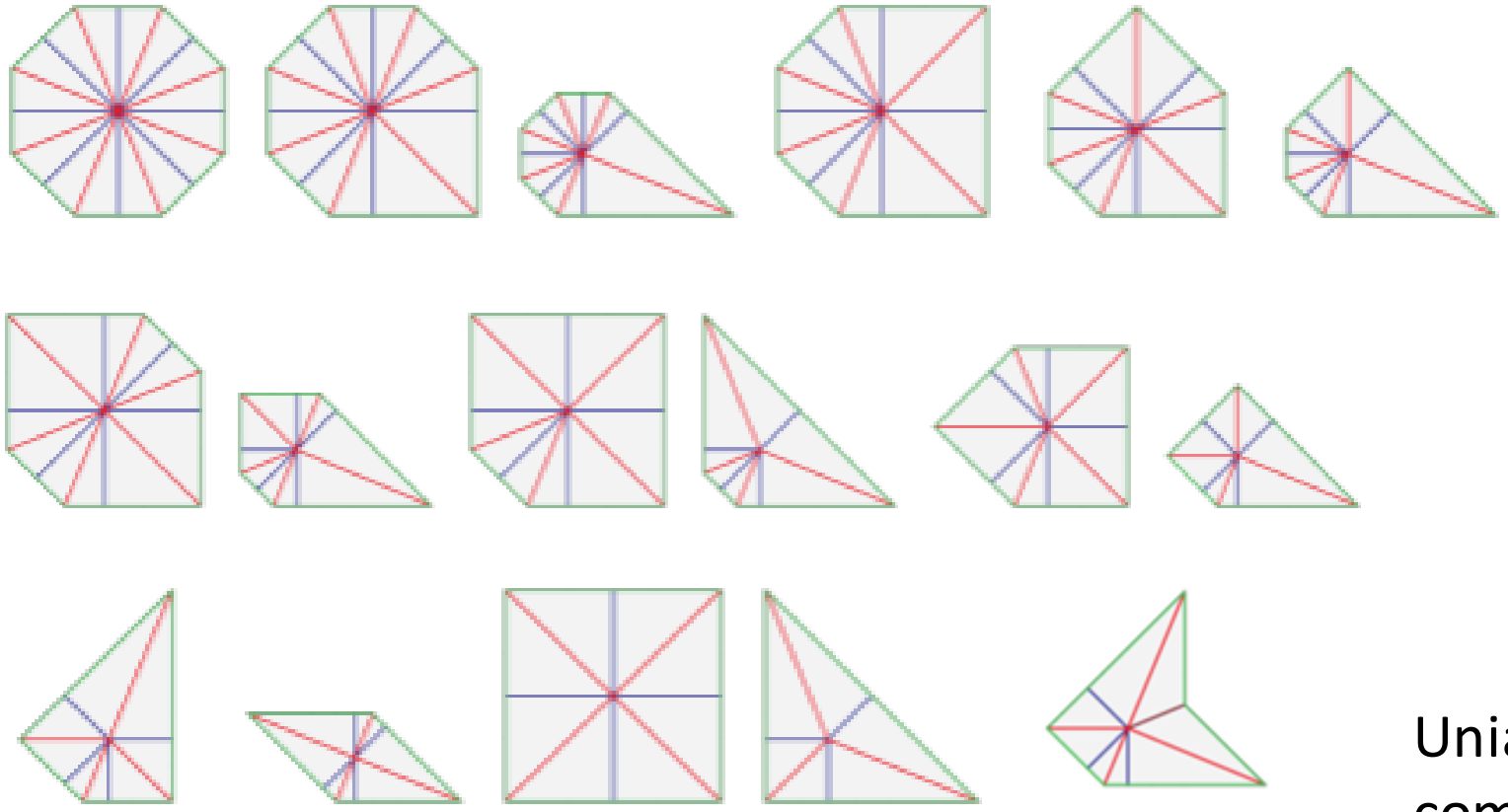
Molecules and tiling (different than tessellations)



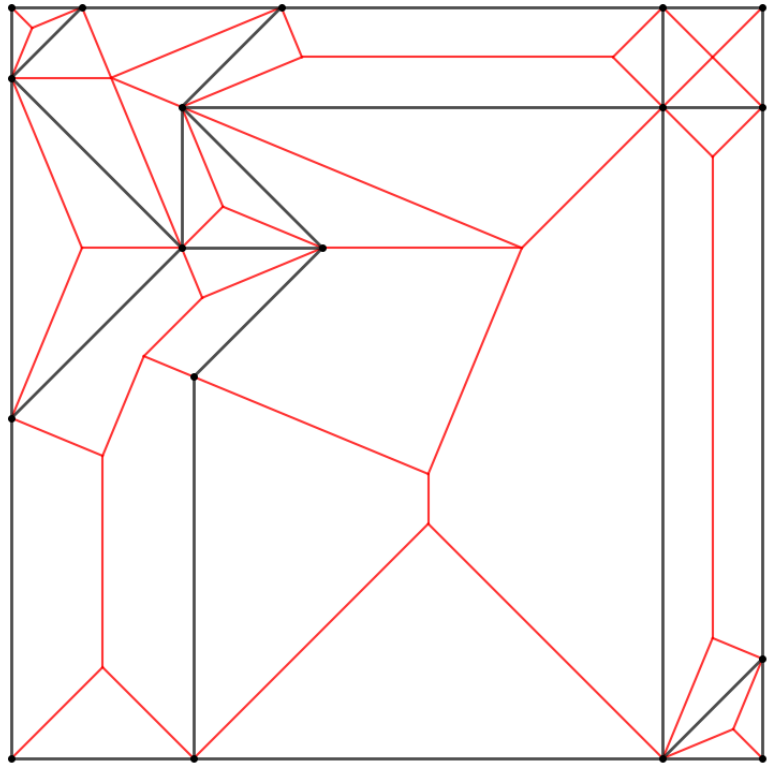


… and many more

Molecules and tiling (different than tessellations)

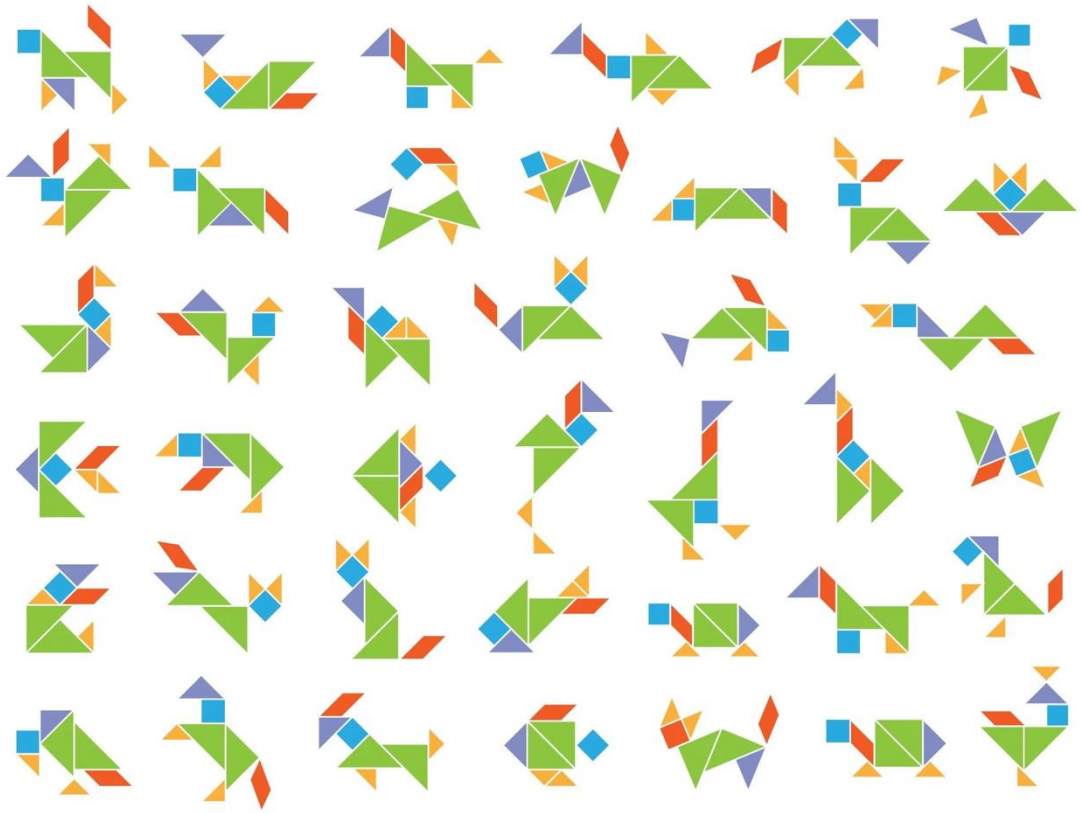


… and many more



Uniaxial crease patterns are combinations of simple molecules

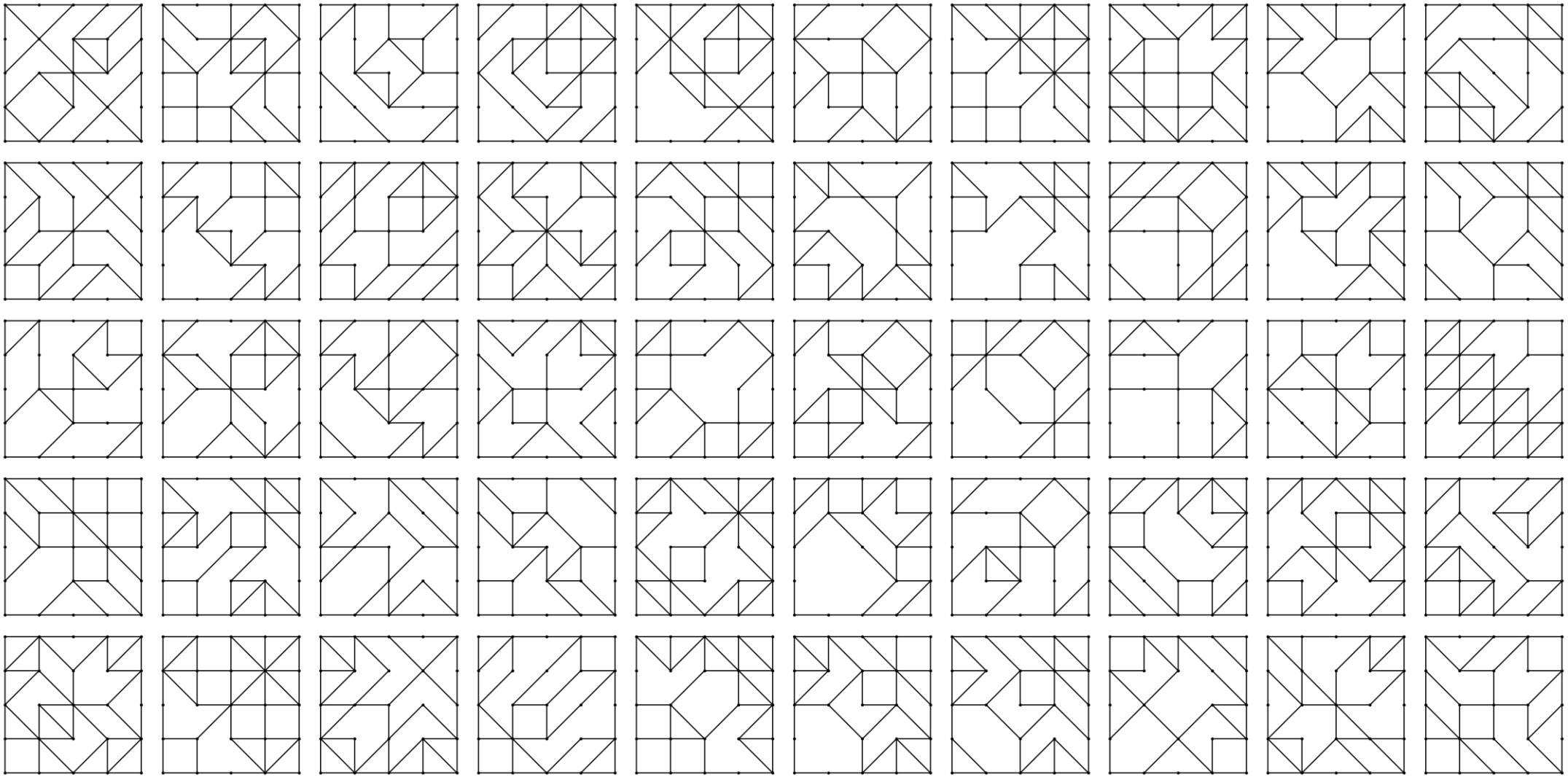
How do we find (all?) combinations of tiles?



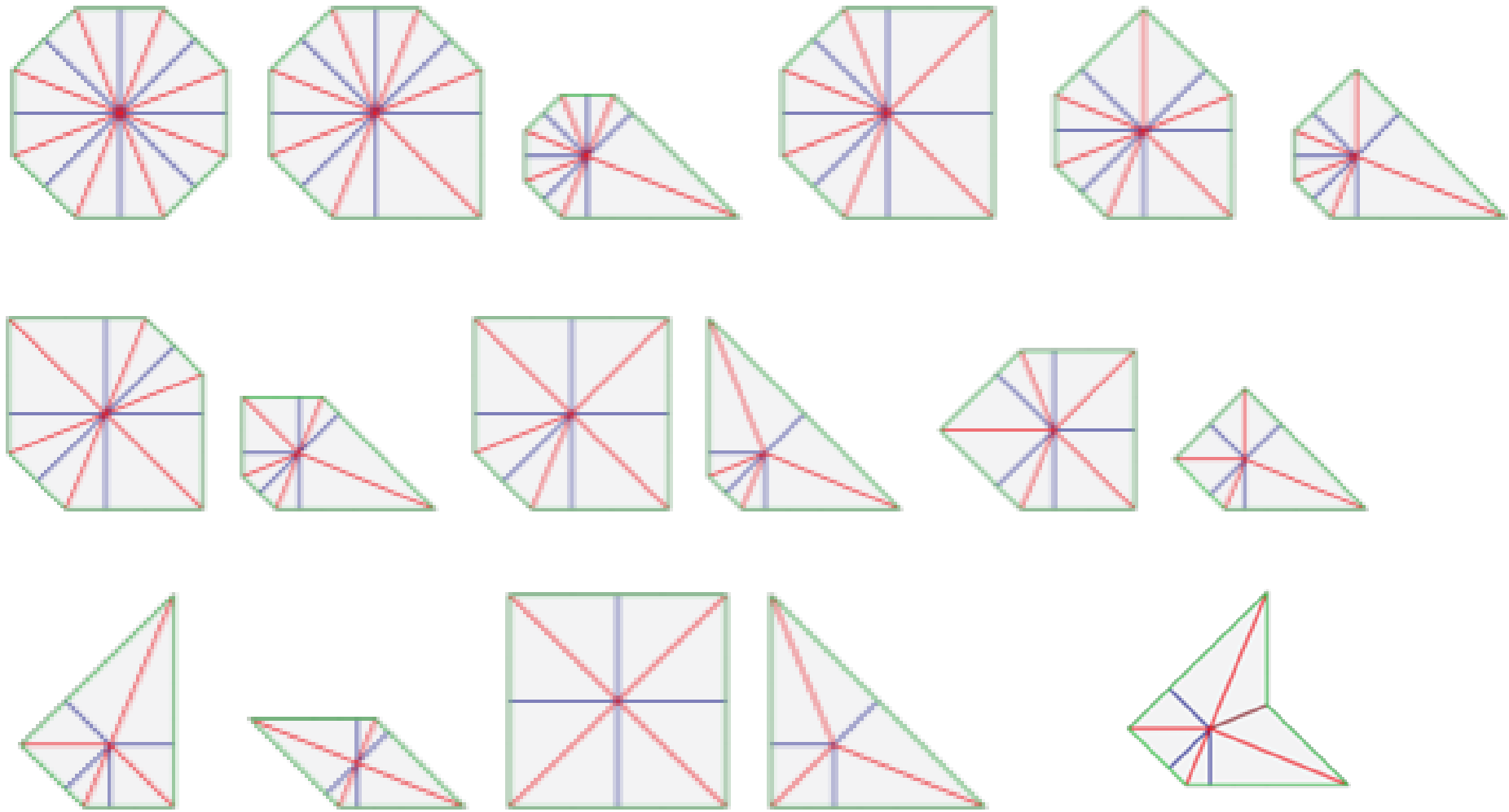
Like tangram enumeration, but:

- Need to form a perfect square
- Tiles can change size
- Many more valid shapes, including concave

How do we find (all?) combinations of tiles?

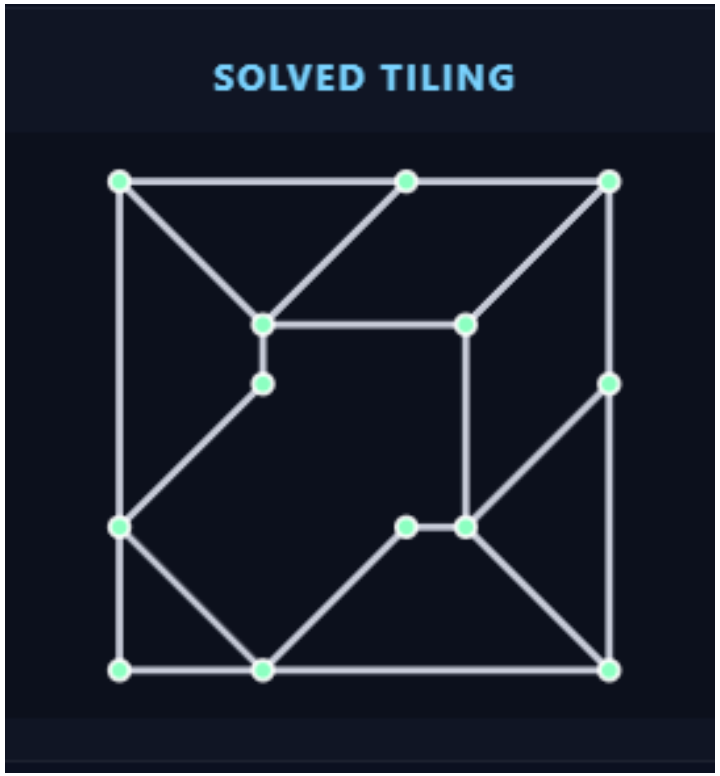


Observation: All axial creases are at 45 degree angles

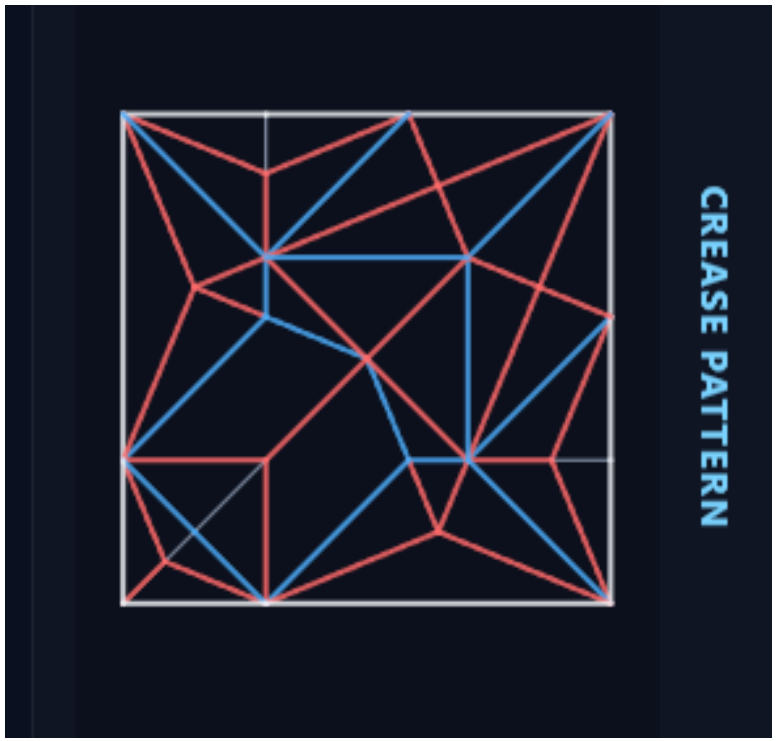


Idea: any network of 45 degree creases will have a corresponding tiling

“Random” network



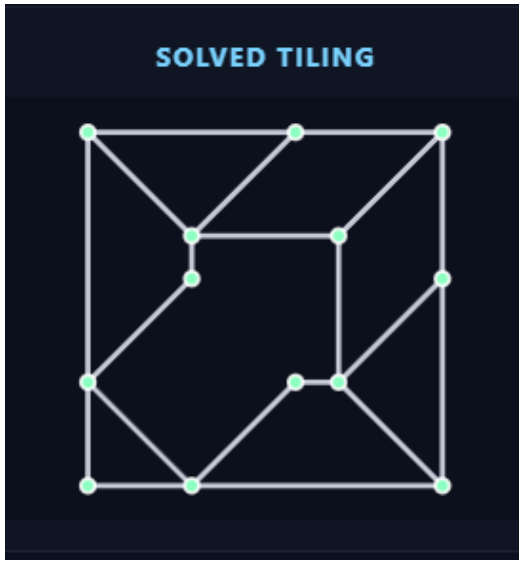
Slightly adjust lengths



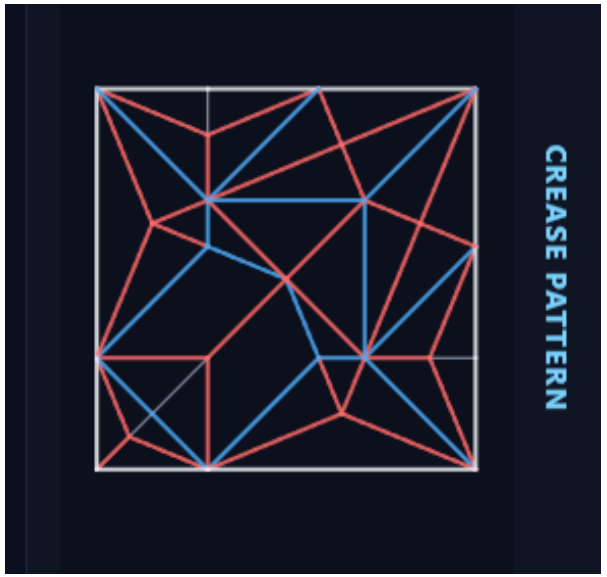
Fill in molecules

Step 1: generate topologies (grid networks)

“Random” network

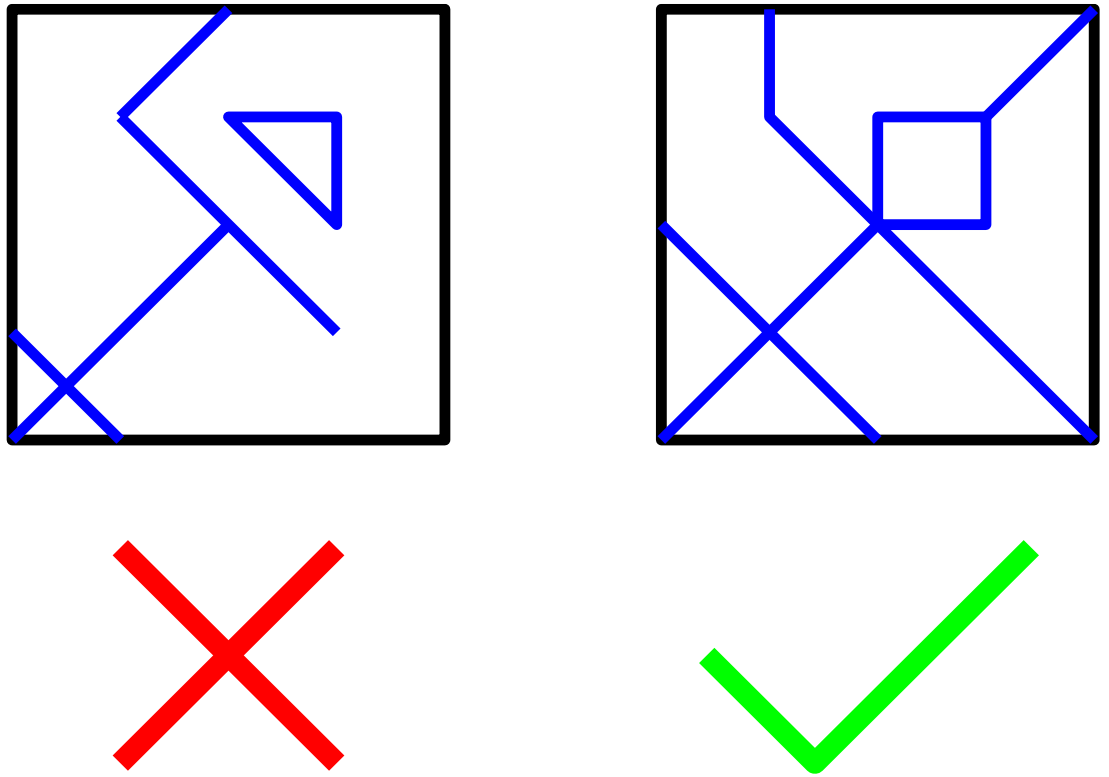


Slightly adjust lengths

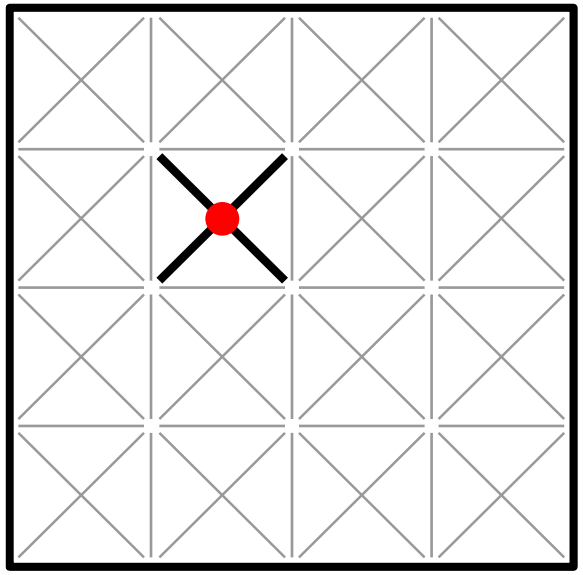


Fill in molecules

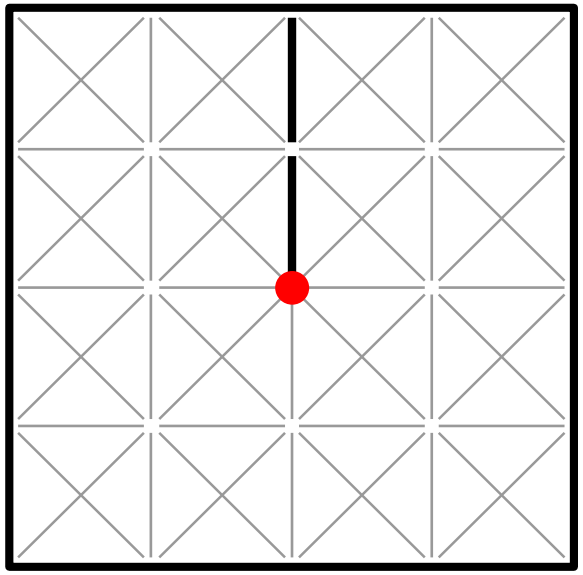
But… not just any random lines



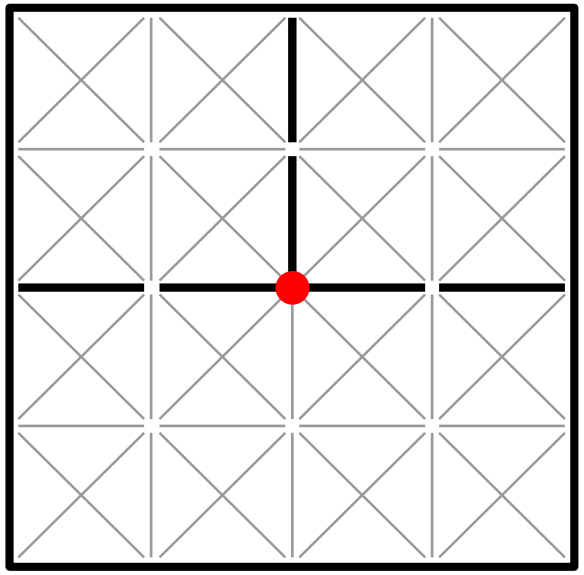
Local topology constraints



No crossing diagonals
in the same cell



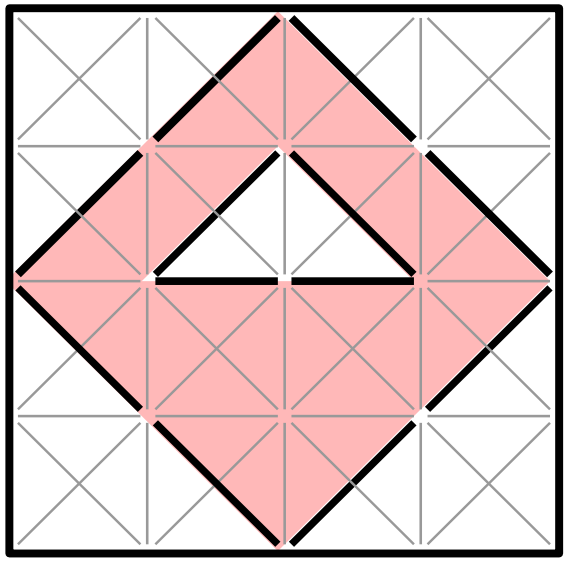
No dangling edges
(degree 1 vertices)



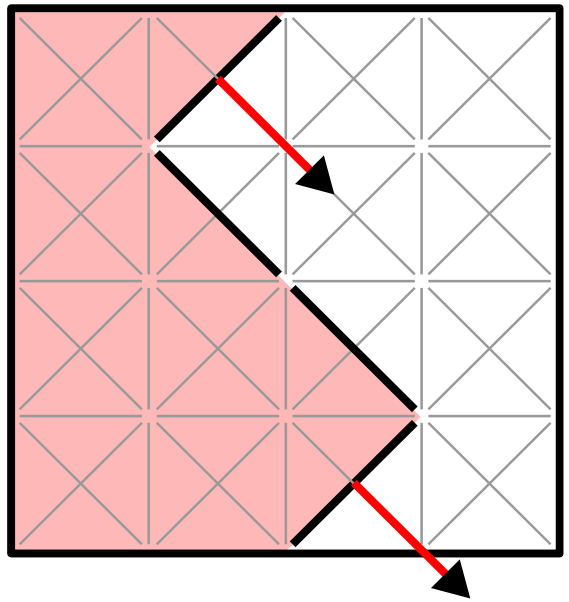
No T junctions

We don’t need ALL cps, just UNLIMITED.
So we apply constraints that make things easier later on

Global topology constraints



No disconnected components
(floating islands)



No duplicate normal vectors
in a polygon

We don’t need ALL cps, just UNLIMITED.
So we apply constraints that make things easier later on

Global topology constraints

No disco ormal vectors
(floating

We don't need A

So we apply con

Satisfiability (SAT) solver formulation

Use a SAT solver: a program that solves logic puzzles

Riddle For Genius – Which Day Is The Thief Telling the Truth?

A thief lies on 6 days of the week. But on 1 day—the same day each week—he only speaks the truth. On 3 consecutive days, he says:

Day 1: I lie on Monday and Tuesday.

Day 2: It's now Thursday, Saturday or Sunday.

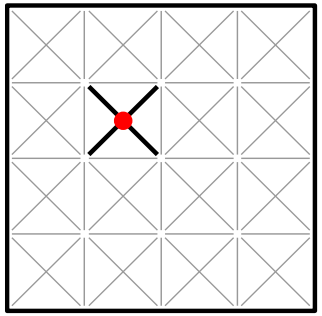
Day 3: I lie on Wednesday and Friday.

SAT solver formulation:

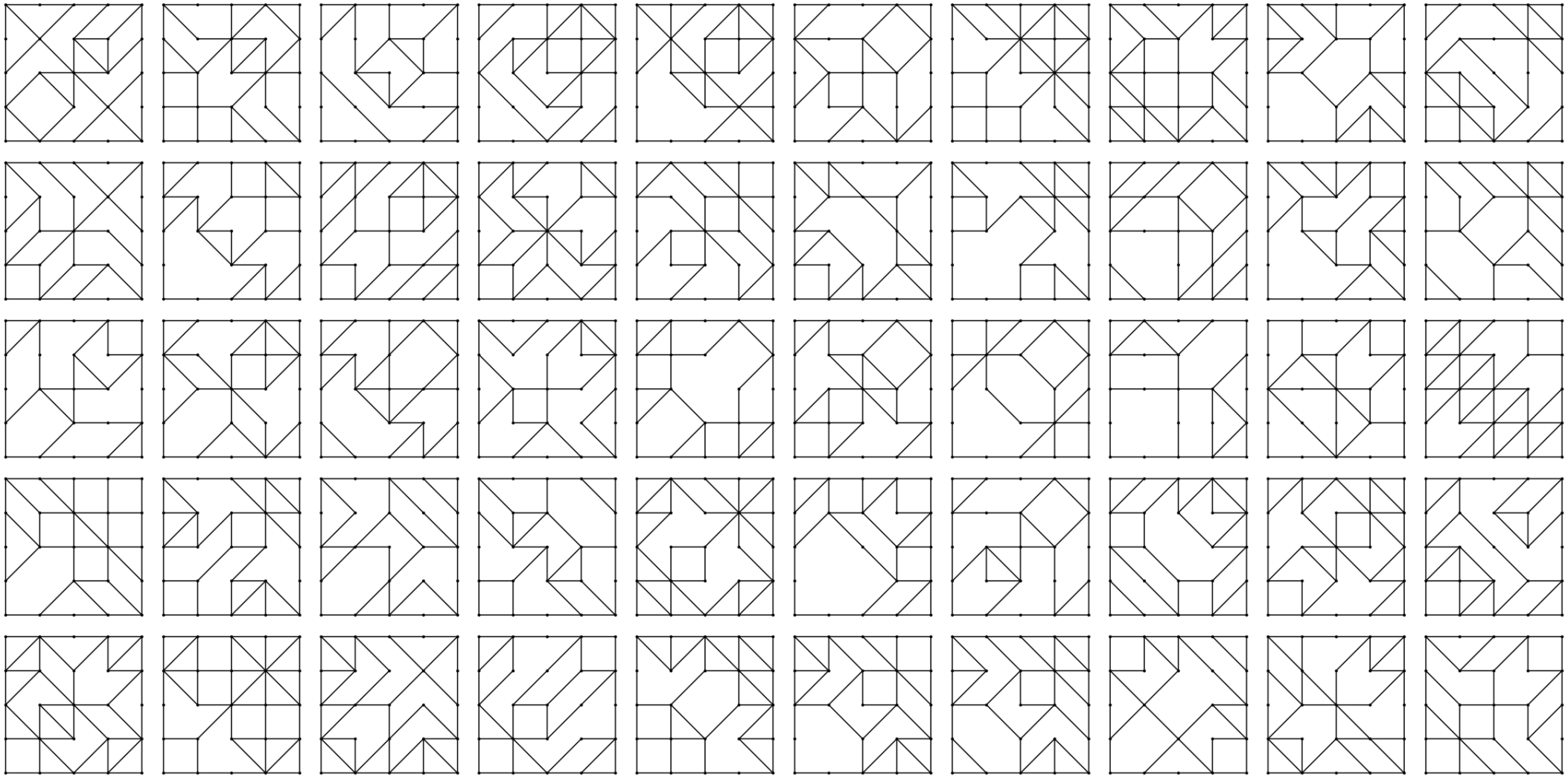
- Each line segment of the grid is a Boolean (true/false) variable
- Formulate constraints as Boolean expressions
- Off-the-shelf SAT solver will exhaust all solutions

$$(\overline{d1_{x,y}} + \overline{d2_{x,y}})$$

No crossing diagonals in the same cell

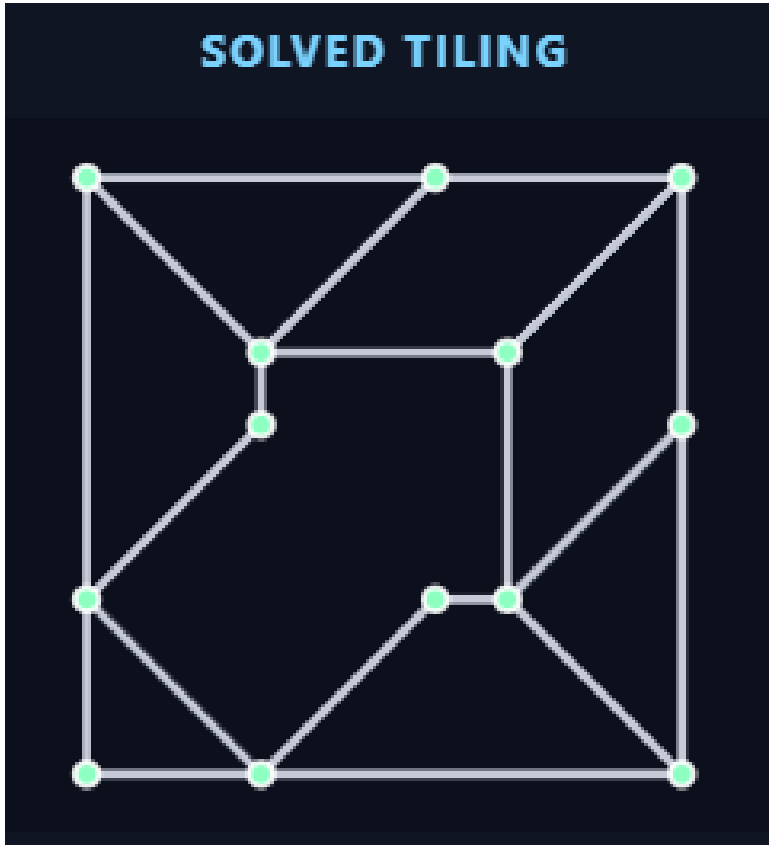


Result: enumerate valid 45 degree topologies

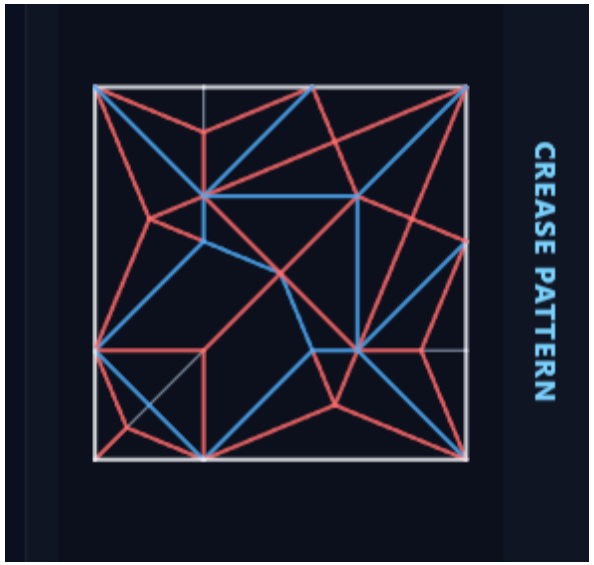


Step 2: adjust geometry

“Random” network

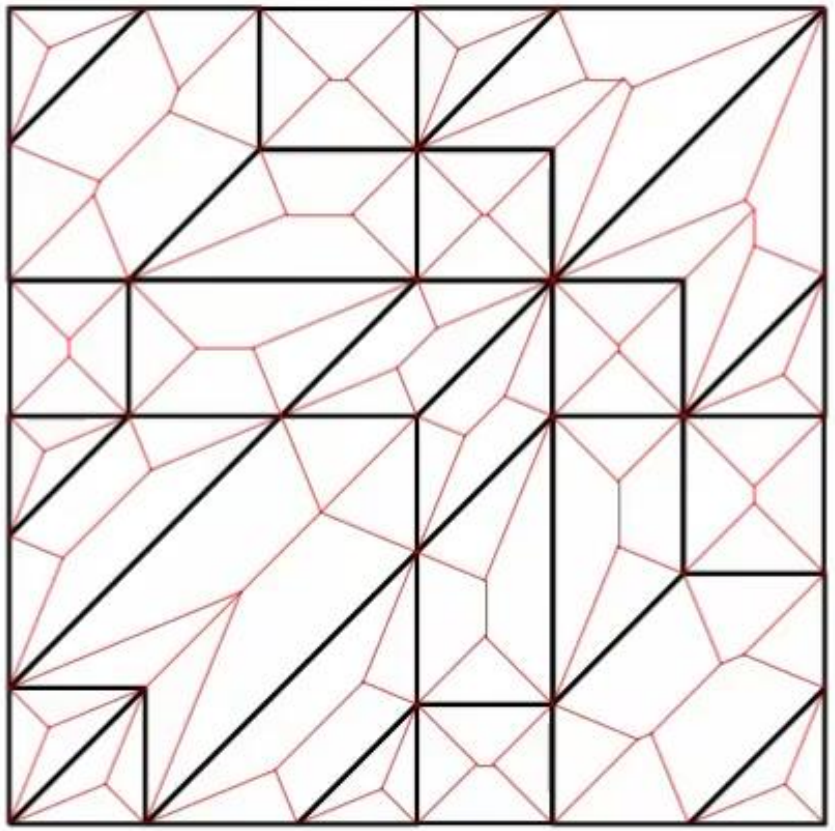


Slightly adjust lengths



Fill in molecules

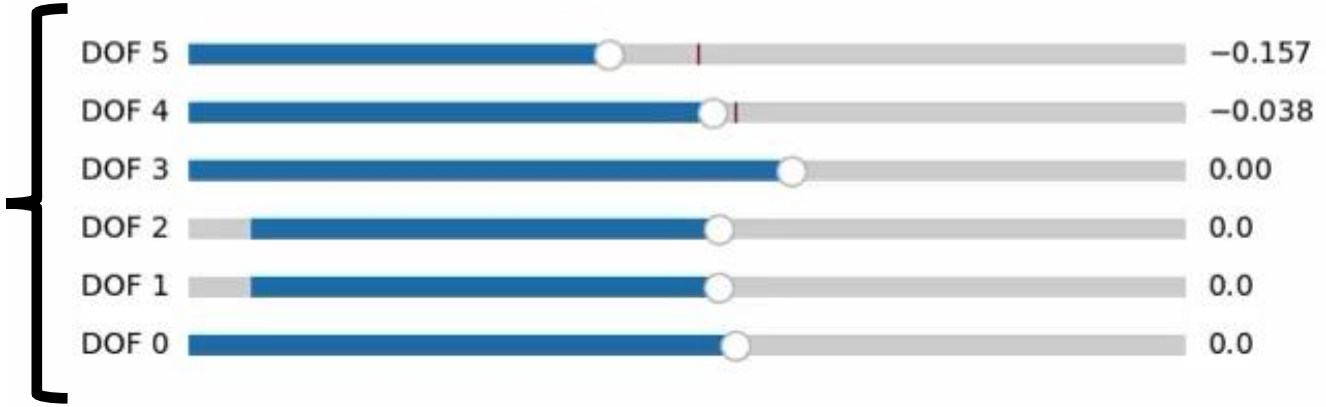
Geometry modification



The cp will be happy, as long as:

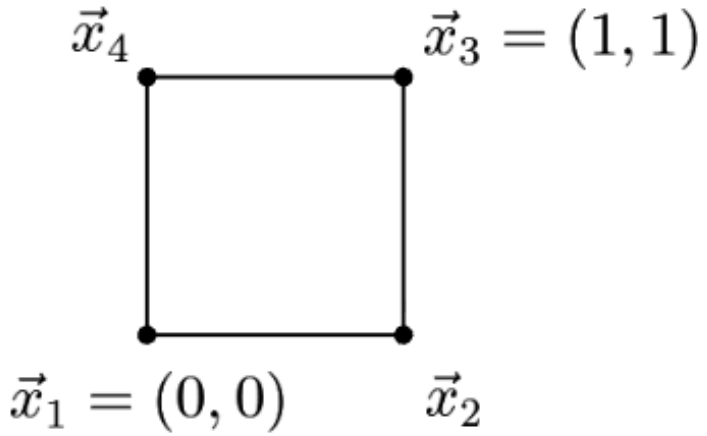
- The edges stay at 45 degrees
- No zero/negative lengths
- No intersections

Want to “lock” lengths together until all degrees of freedom are constrained (one solution)



Computing degrees of freedom

Formulate a constraint matrix. Ex:



Edge angle constraints

Boundary pinning

Diagonal symmetry

$$\underbrace{\left[\begin{array}{cccccccc} 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ \hdashline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hdashline 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \end{array}\right]}_{A} \underbrace{\left[\begin{array}{c} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{array}\right]}_{\vec{x}} = \underbrace{\left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right]}_{\vec{b}}$$

The matrix describes constraints in movement. Each row is a different “rule”

Computing degrees of freedom

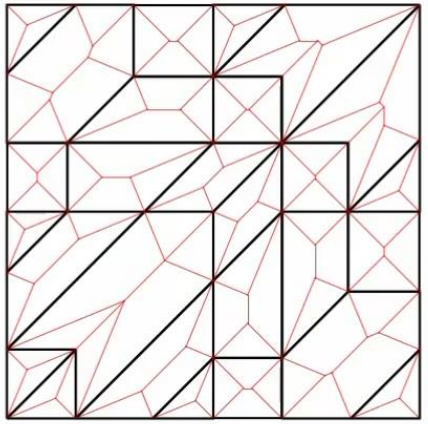
Each column of the null space of the constraint matrix is a degree of freedom

Edge angle constraints

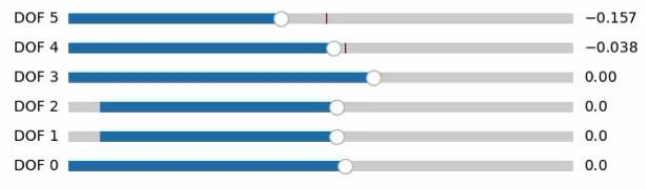
Boundary pinning

Diagonal symmetry

$$\underbrace{\left[\begin{array}{cccccccc} 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ \hdashline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hdashline 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \end{array}\right]}_{A} \underbrace{\begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\vec{b}}$$

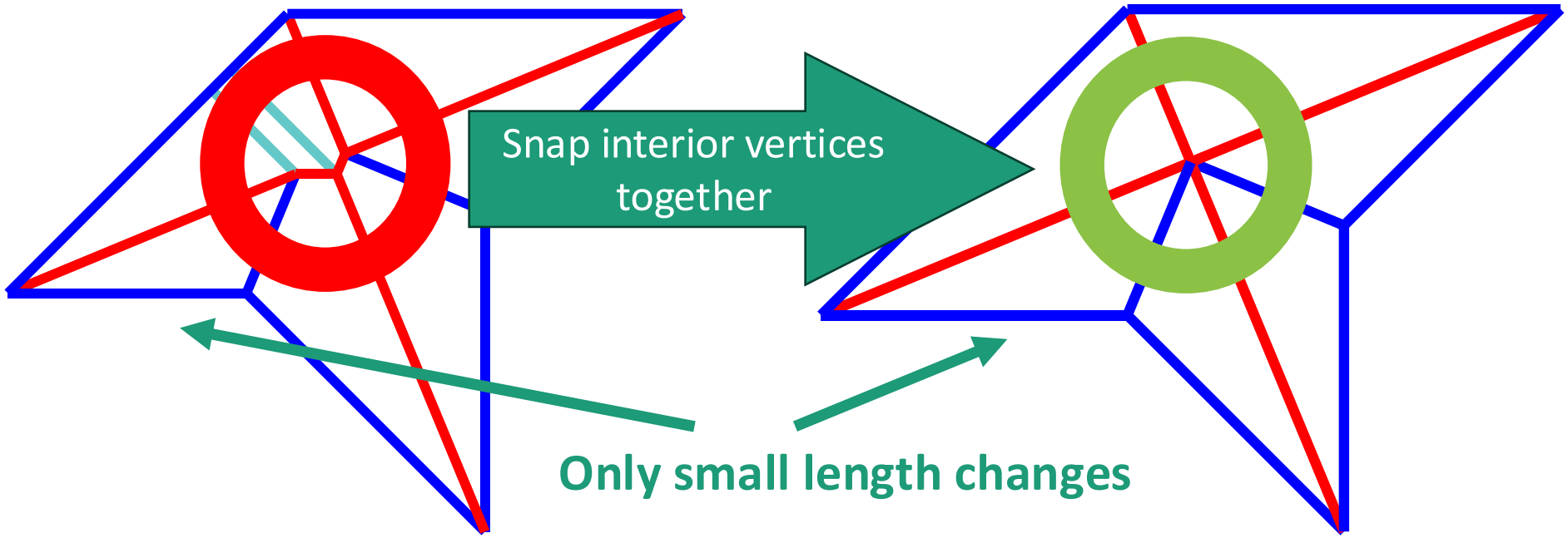


$$\text{null}(A)$$



The null space describes allowable motions within the constraints

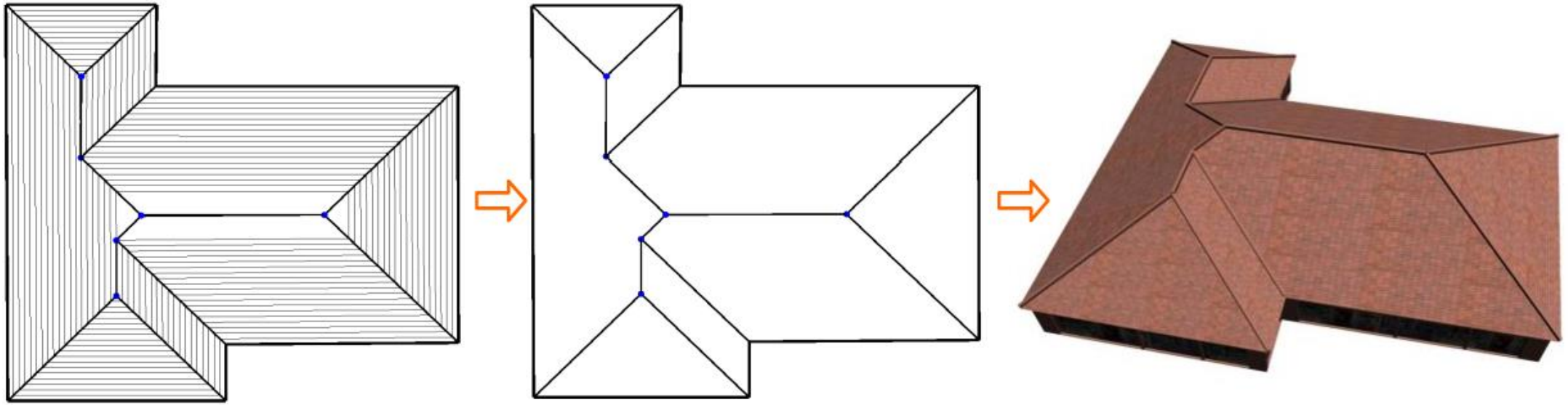
Goal: make crease pattern “nice” to fold



Messy/unpleasant to fold

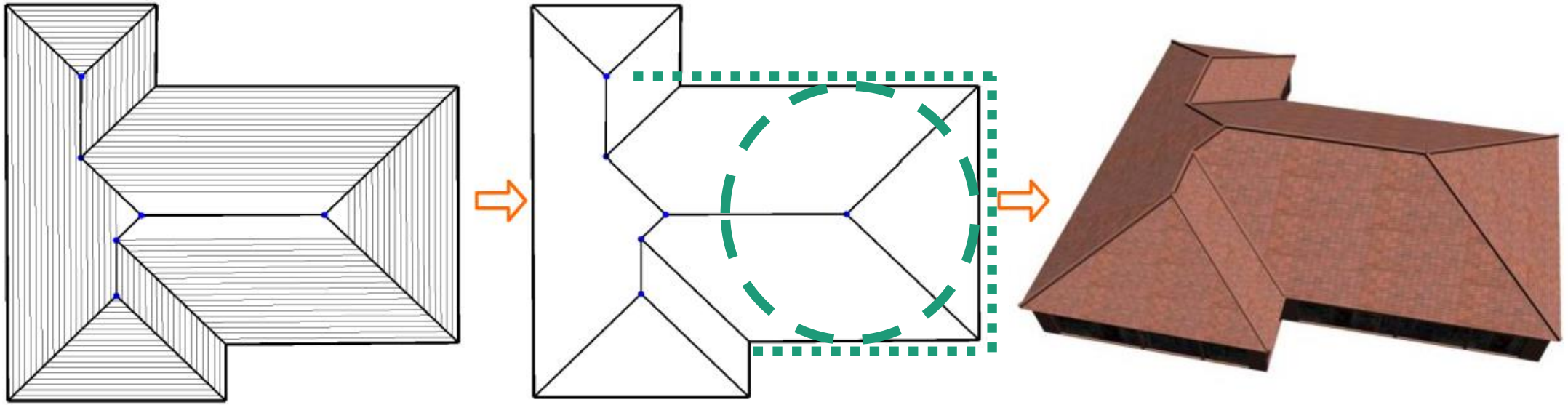
Nice

Straight skeletons



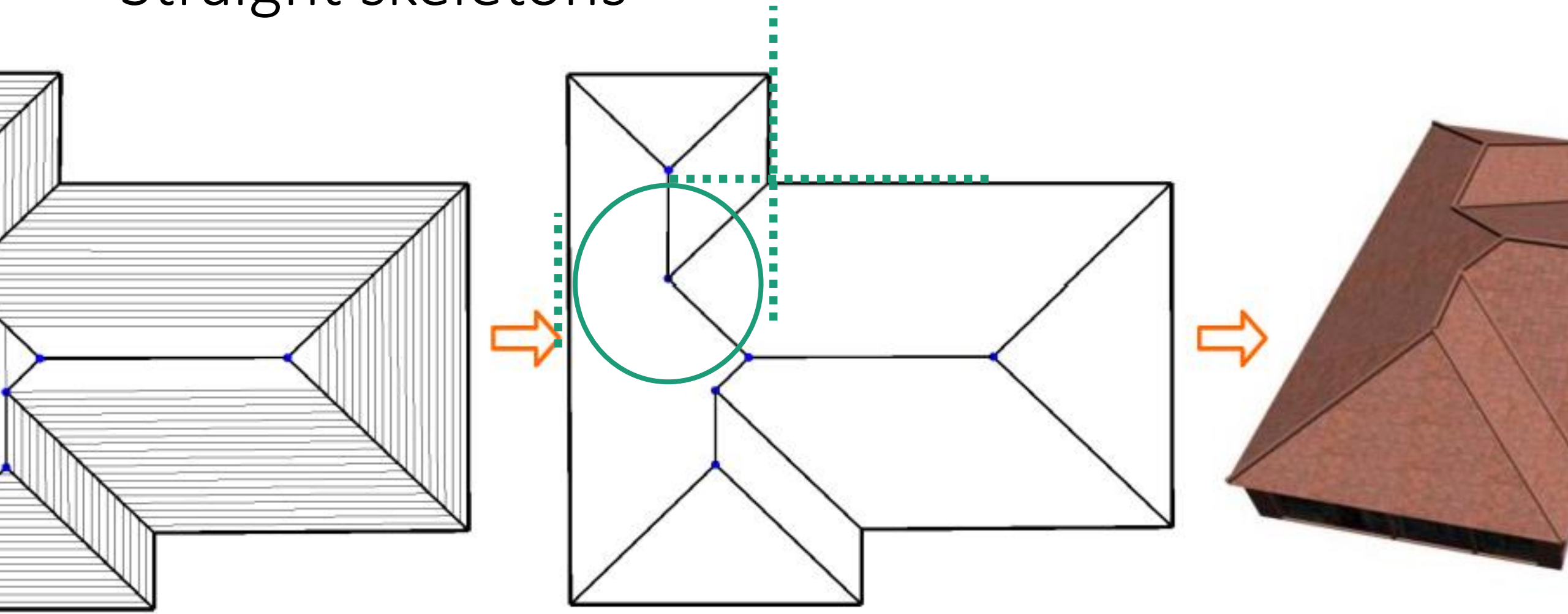
This is the straight skeleton
of the outer polygon

Straight skeletons



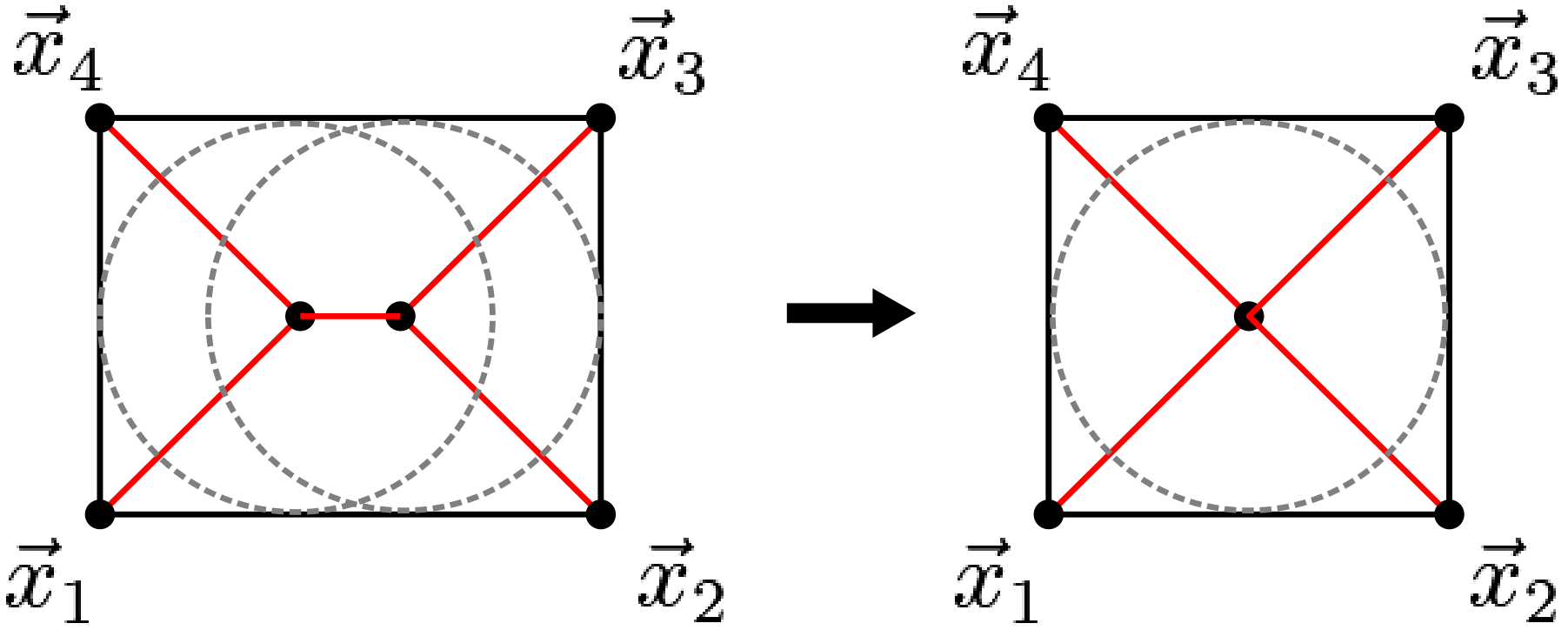
A point in the straight skeleton is the center of an incircle tangent to 3 or more sides

Straight skeletons



Lines don’t have to actually touch the circle

To collapse interior vertices, force 4 sides to be tangent



Quadruplet: a set of 4 sides that we force to share a common circle

Constraint formulation for a quadruplet

To make the constraint matrix full rank,
we can add more rows

The new row in the matrix is: $$\sum_{i=1}^{4}((w_i n_{ix})x_i + (w_i n_{iy})y_i) = 0$$

Where w is in the left null space of Q:

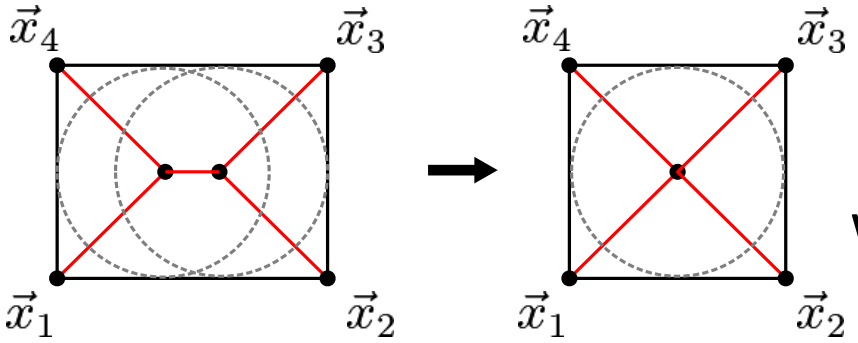
(For derivation, ask me later)

$$\underbrace{\begin{bmatrix} n_{1x} & n_{1y} & -1 \\ n_{2x} & n_{2y} & -1 \\ n_{3x} & n_{3y} & -1 \\ n_{4x} & n_{4y} & -1 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} p_x \\ p_y \\ r \end{bmatrix}}_{\vec{\xi}} = \underbrace{\begin{bmatrix} \hat{n}_1 \cdot \vec{v}_1 \\ \hat{n}_2 \cdot \vec{v}_2 \\ \hat{n}_3 \cdot \vec{v}_3 \\ \hat{n}_4 \cdot \vec{v}_4 \end{bmatrix}}_{\vec{w}}$$

Constraining 4 sides requires 1 new constraint

Constraint formulation for a quadruplet

Ex: to constrain a rectangle into a square:



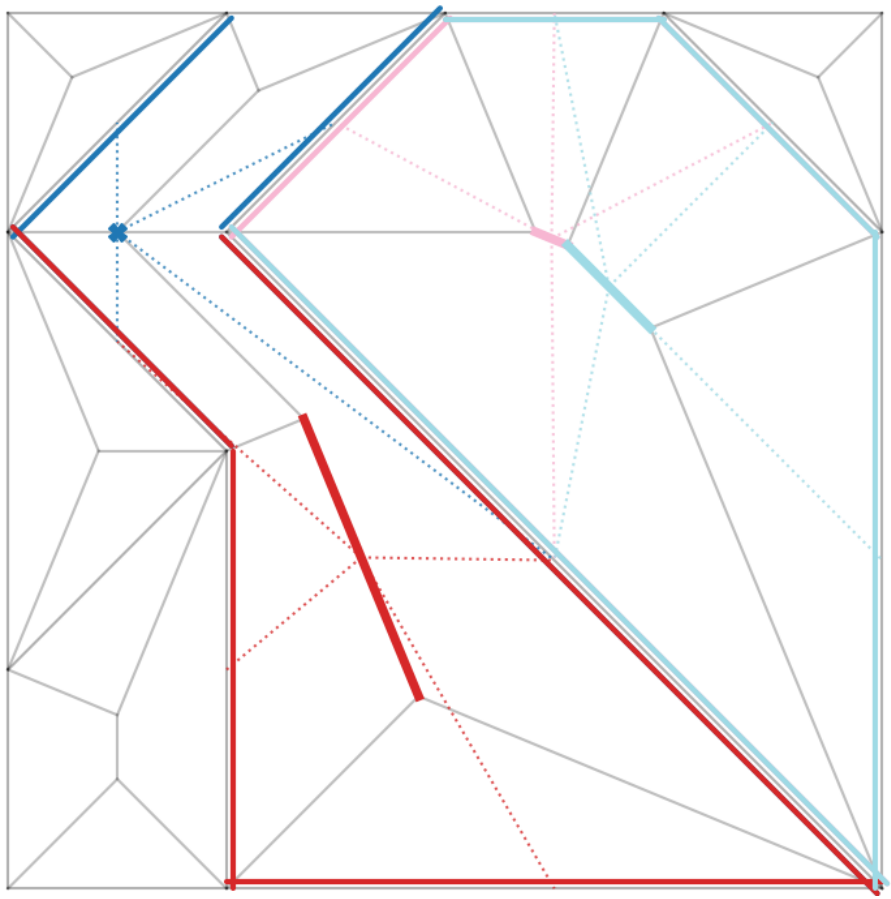
$$\begin{bmatrix} -1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix}$$

Which is equivalent to:

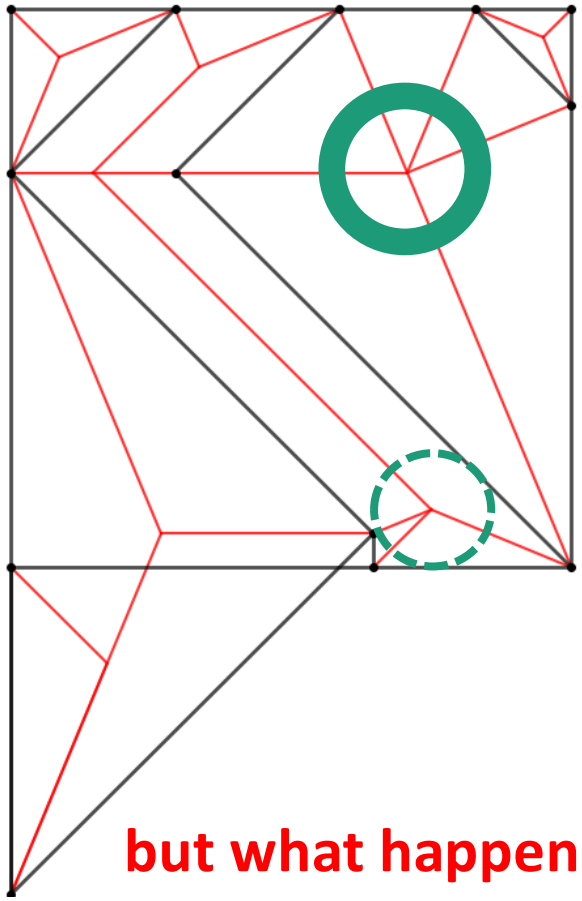
$$x_3 - x_1 = y_3 - y_1$$

In this case, the constraint equation is essentially ”length = width”

Choosing quadruplets is not easy



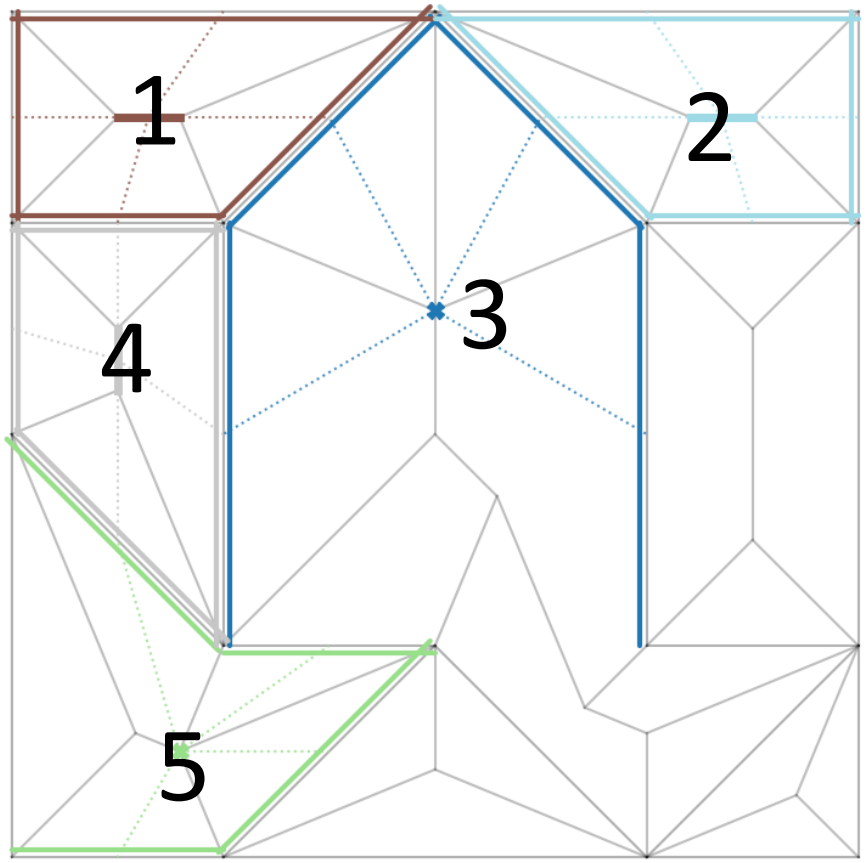
Double success!



but what happened here…

Problem is unrelated to quadruplet

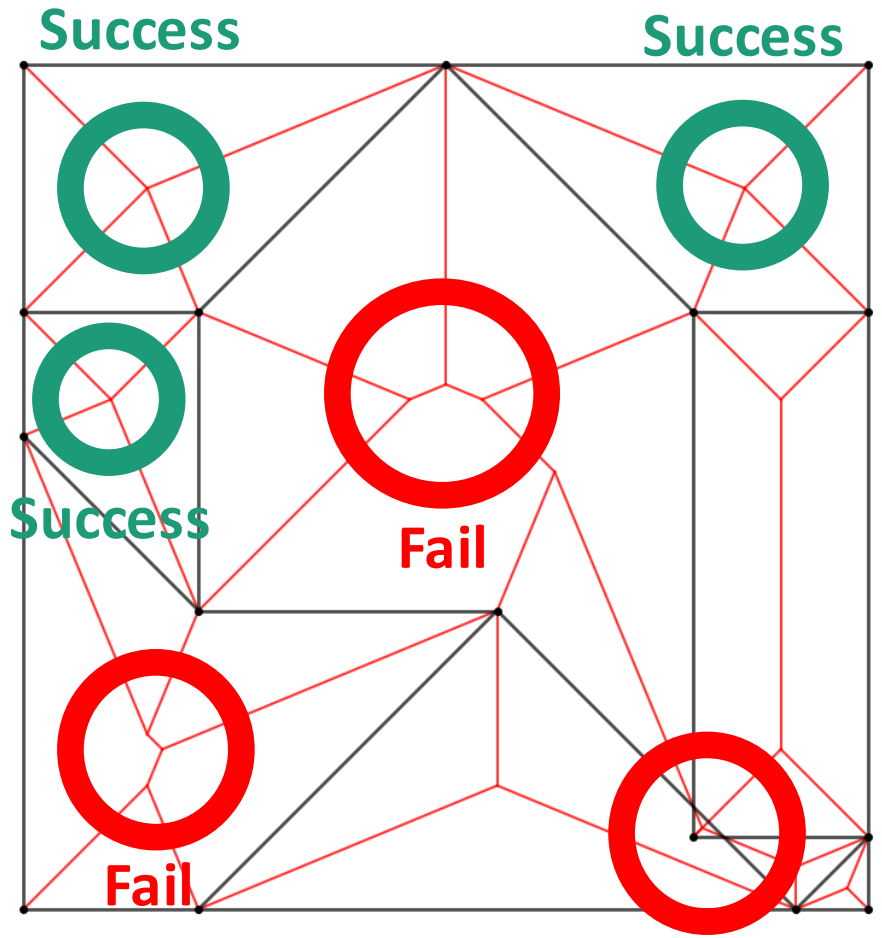
Choosing quadruplets is not easy



Attempted quadruplet constraints

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

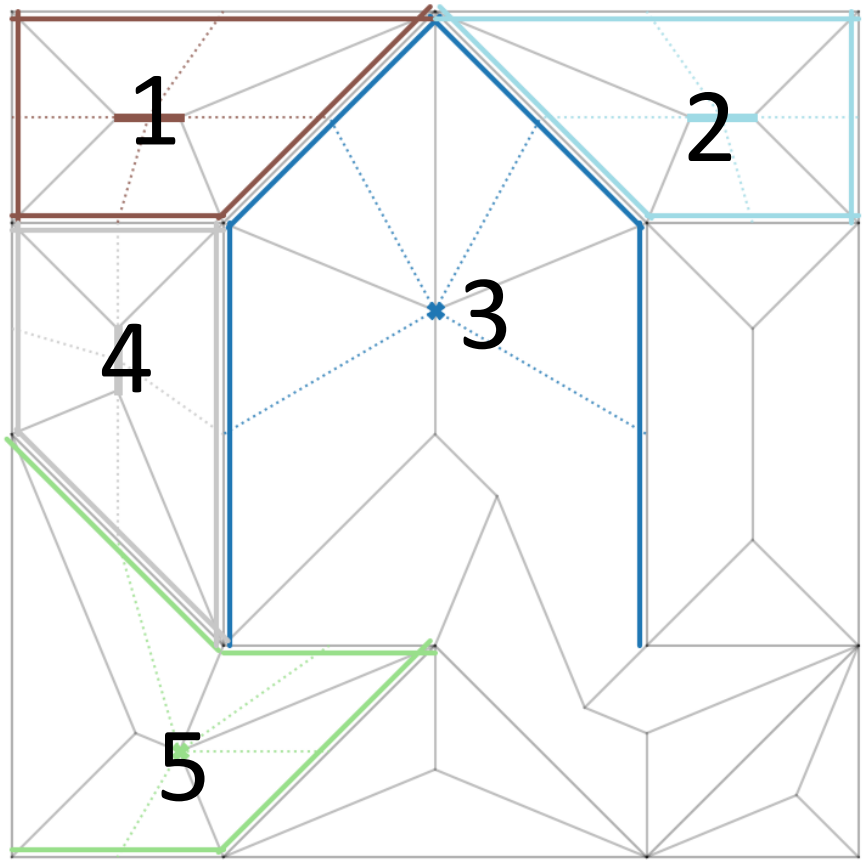
Solve



Results

Additional problem

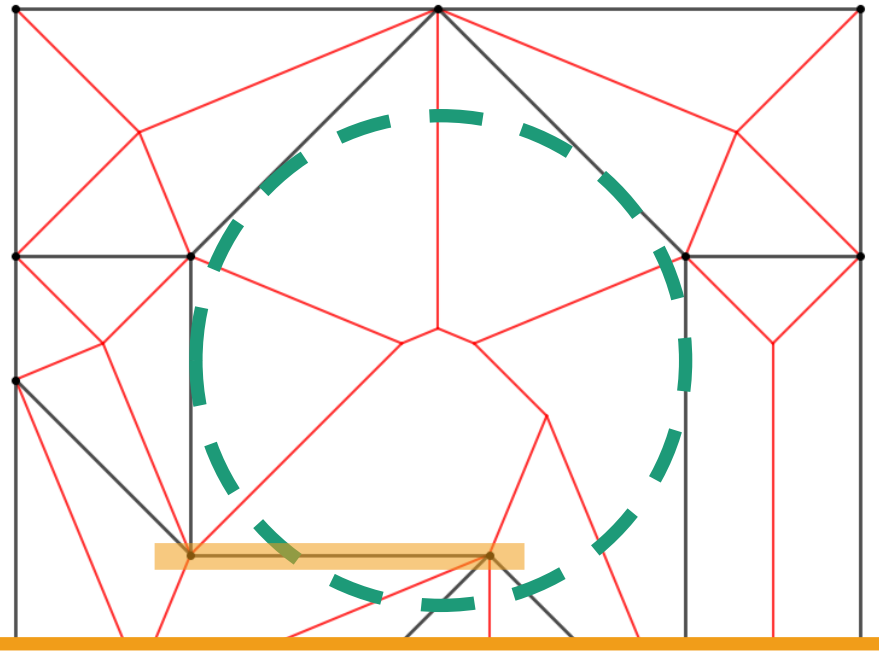
Choosing quadruplets is not easy



Attempted quadruplet constraints

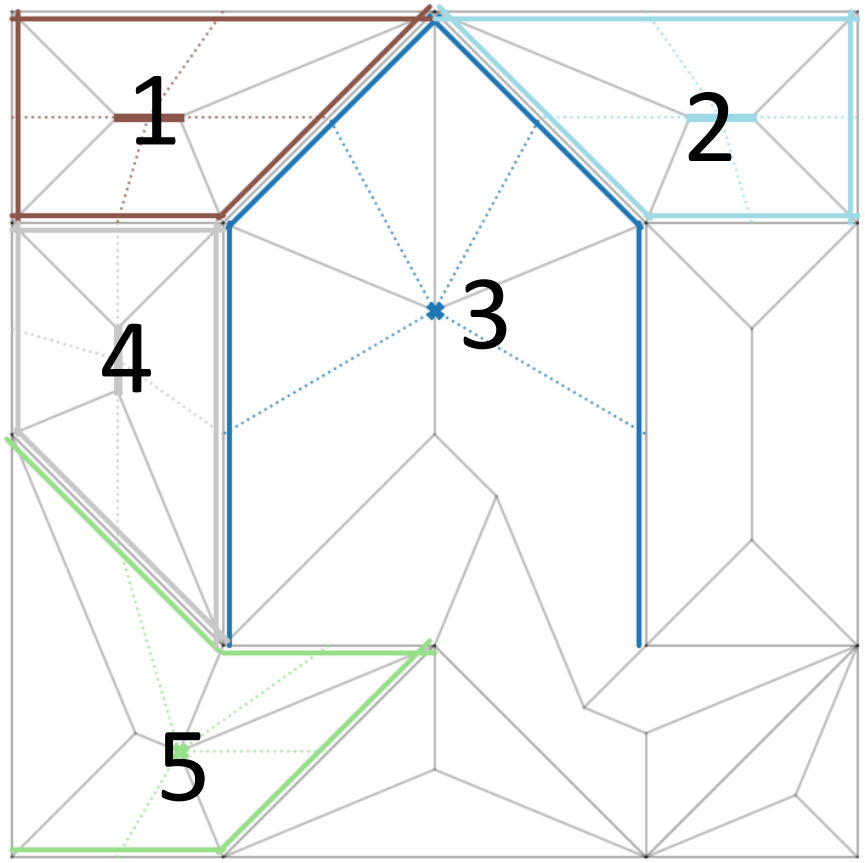
$A\vec{x} = \vec{b}$

Solve

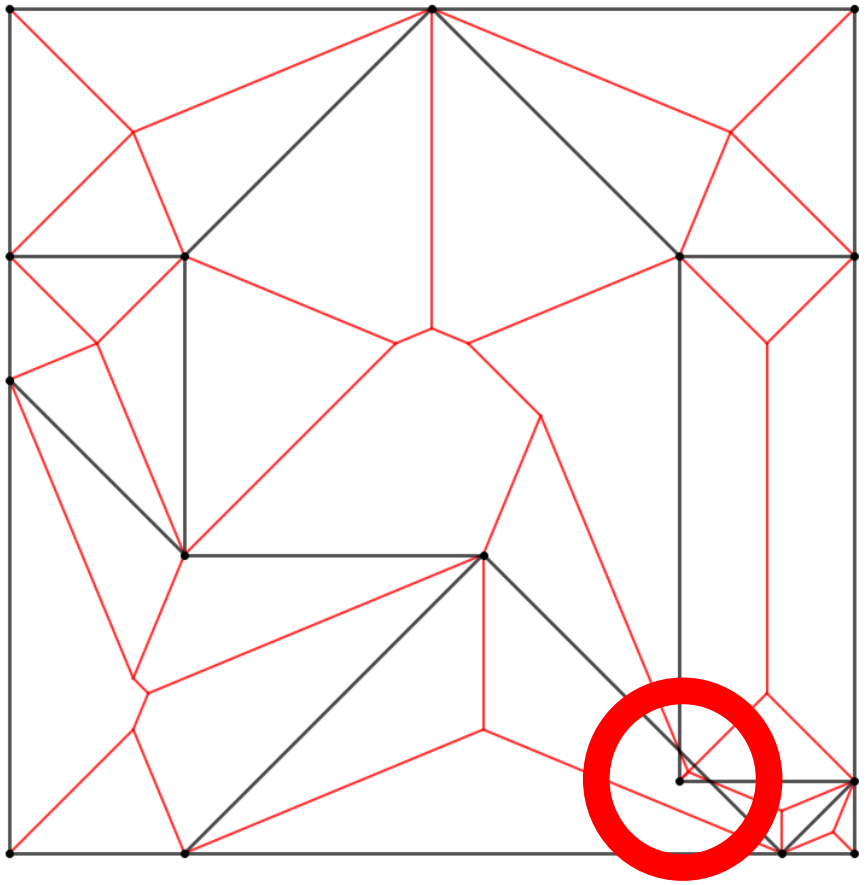


The quadruplet was satisfied, the problem came from an incircle invasion by another edge

Choosing quadruplets is not easy

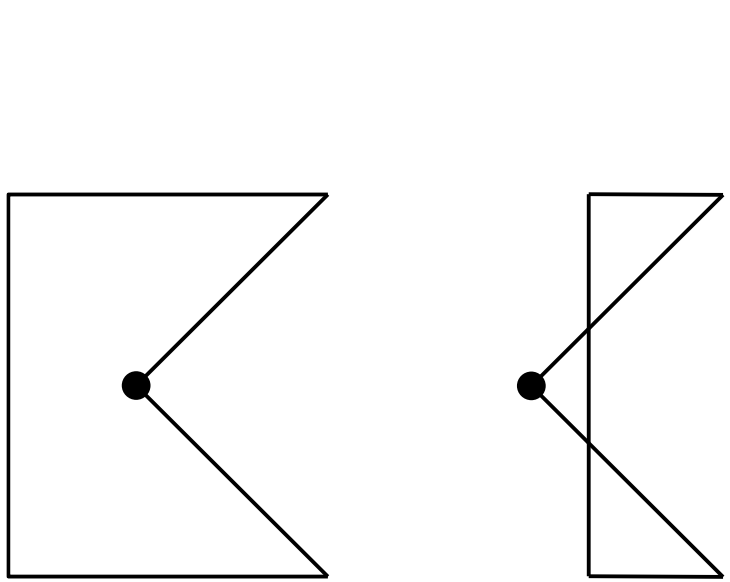


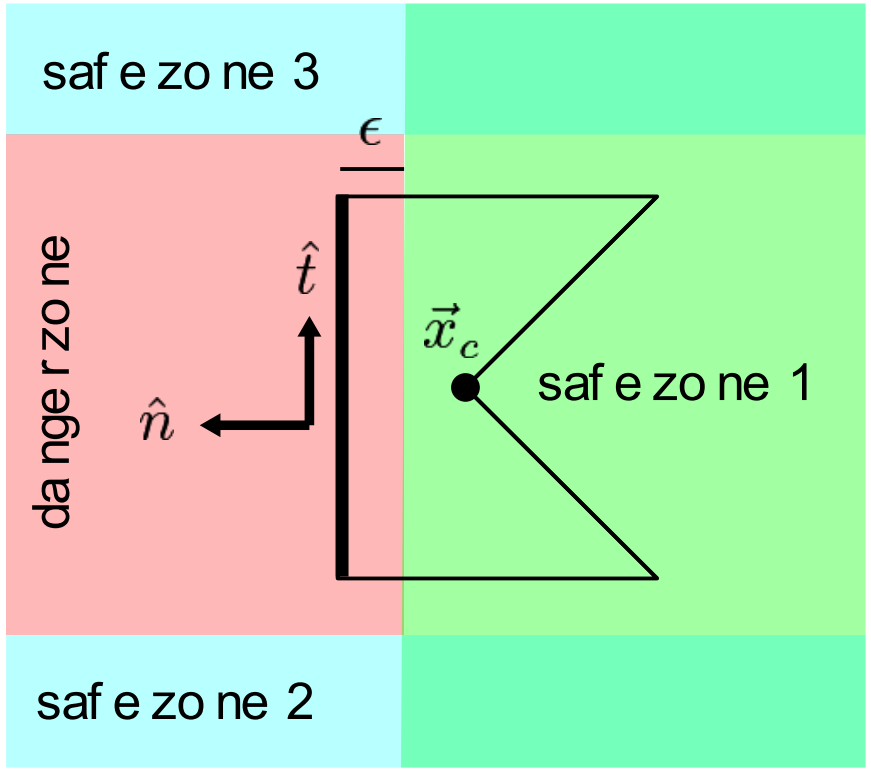
Attempted quadruplet constraints



This is also unrelated to quadruplets

Inequality constraint: bowtie clipping





Keep concave vertices out of danger zones

Mixed Integer Linear Programming (MILP)

Mixed Integer Linear Programming (MILP) is a branch of math used in operations research:

- Supply chain
- Transportation
- Manufacturing

etc

MILP can handle inequality constraints

MILP formulation

Define a compound vector:

$$\vec{X} = \left[\vec{x},\ \vec{z}_{quads},\ \vec{p}_{incircles},\ \vec{r}_{incircles},\ \vec{b}_{bowtie} \right]^T$$

Vertex positions

Quadruplet activation

Extra variables

$$\max_{\vec{X}} \quad \vec{c}^T \vec{X}$$

Objective function: activate as many quadruplets as possible

$$\text{subject to} \quad M\vec{X} \leq \vec{B}$$

Encode inequality constraints into big constraint matrix M

Solver will make the cp as “nice” as possible until constraints are violated

MILP formulation

$$
\begin{array}{ll}
\text{Basic constraints} & \\
\text{Edge Integrity} & \\
\text{Quadruplets} & \\
 & \\
\text{Incircle Clearance} & \\
 & \\
\text{Anti-bowtie} &
\end{array}
\underbrace{\begin{bmatrix}
A & 0 & 0 & 0 & 0 \\
M_{edges} & 0 & 0 & 0 & 0 \\
-N_{quad} & C \cdot I_{quad} & N_{incircle} & -I_{quad} & 0 \\
N_{quad} & C \cdot I_{quad} & -N_{incircle} & I_{quad} & 0 \\
N_{invader} & C \cdot I_{inv} & -N_{incircle} & -I_{inv} & 0 \\
0 & 0 & 0 & 0 & -S_{sector} \\
N_{bowtie} & 0 & 0 & 0 & C \cdot I_{bowtie}
\end{bmatrix}}_{M}
\underbrace{\begin{bmatrix}
\vec{x} \\
\vec{z}_{quads} \\
\vec{P}_{incircles} \\
\vec{r}_{incircles} \\
\vec{b}_{bowtie}
\end{bmatrix}}_{\vec{X}}
\leq
\underbrace{\begin{bmatrix}
\mathbf{0} \\
-\boldsymbol{\epsilon} \\
\mathbf{C} \\
\mathbf{C} \\
\mathbf{C} \\
-\mathbf{1} \\
\mathbf{C} - \boldsymbol{\epsilon}
\end{bmatrix}}_{\vec{B}}
$$

(For full derivation, ask me later)

Solver will make the cp as “nice” as possible until constraints are violated

MILP formulation

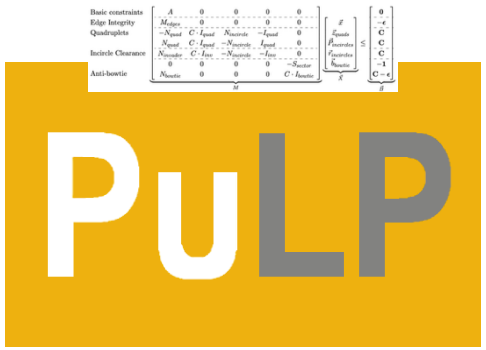
$$
\begin{array}{l}
\text{Basic constraints} \\
\text{Edge Integrity} \\
\text{Quadruplets} \\
\\
\text{Incircle Clearance} \\
\\
\text{Anti-bowtie}
\end{array}
\underbrace{\begin{bmatrix}
A & 0 & 0 & 0 & 0 \\
M_{edges} & 0 & 0 & 0 & 0 \\
-N_{quad} & C \cdot I_{quad} & N_{incircle} & -I_{quad} & 0 \\
N_{quad} & C \cdot I_{quad} & -N_{incircle} & I_{quad} & 0 \\
N_{invader} & C \cdot I_{inv} & -N_{incircle} & -I_{inv} & 0 \\
0 & 0 & 0 & 0 & -S_{sector} \\
N_{bowtie} & 0 & 0 & 0 & C \cdot I_{bowtie}
\end{bmatrix}}_{M}
\underbrace{\begin{bmatrix}
\vec{x} \\
\vec{z}_{quads} \\
\vec{P}_{incircles} \\
\vec{r}_{incircles} \\
\vec{b}_{bowtie}
\end{bmatrix}}_{\vec{X}}
\leq
\underbrace{\begin{bmatrix}
\mathbf{0} \\
-\boldsymbol{\epsilon} \\
\mathbf{C} \\
\mathbf{C} \\
\mathbf{C} \\
-\mathbf{1} \\
\mathbf{C} - \boldsymbol{\epsilon}
\end{bmatrix}}_{\vec{B}}
$$

Plug into solver

PuLP

The problem can be solved by standard off-the-shelf code libraries

MILP formulation



Ideal set of quadruplets

Edge angle constraints

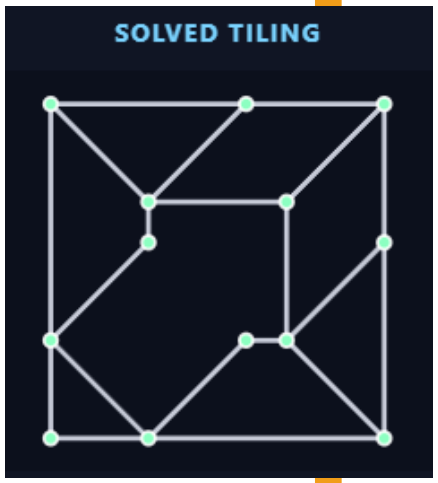
Boundary pinning

Diagonal symmetry

$$\begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \end{bmatrix}_{A} \underbrace{\begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\vec{b}}$$

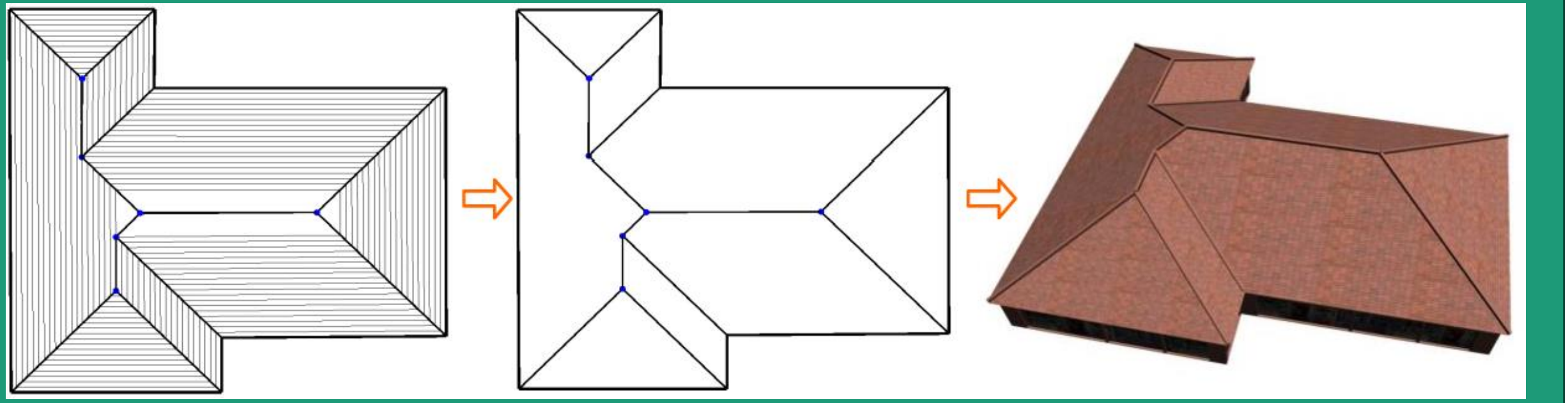
[additional rows from quadruplet constraints]

Solve



Once the quadruplets are selected, we solve and get a tiling that won't slide around

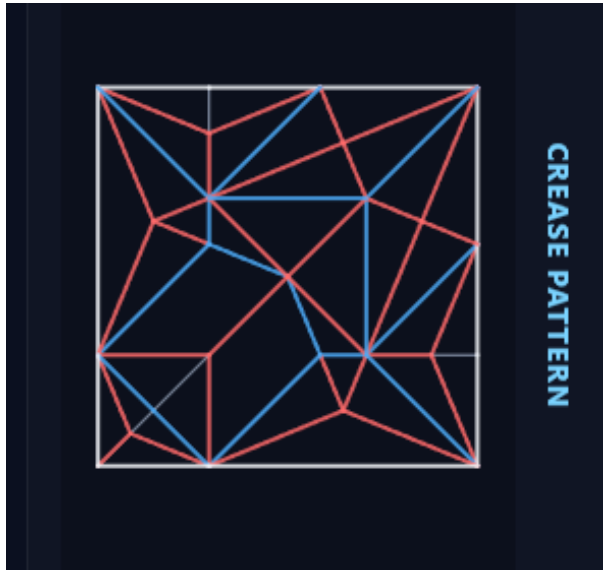
Step 3: Generate crease pattern

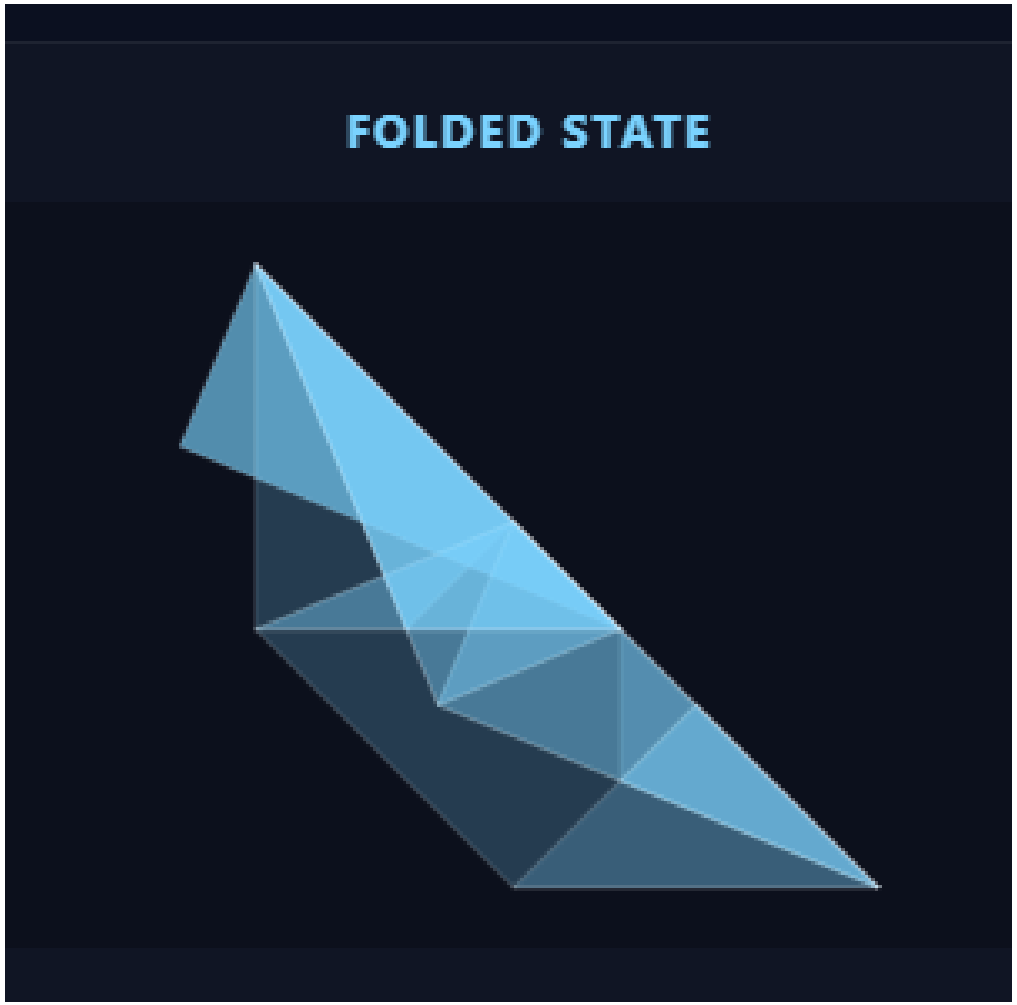


Random network → Slightly adjust lengths

Fill in molecules

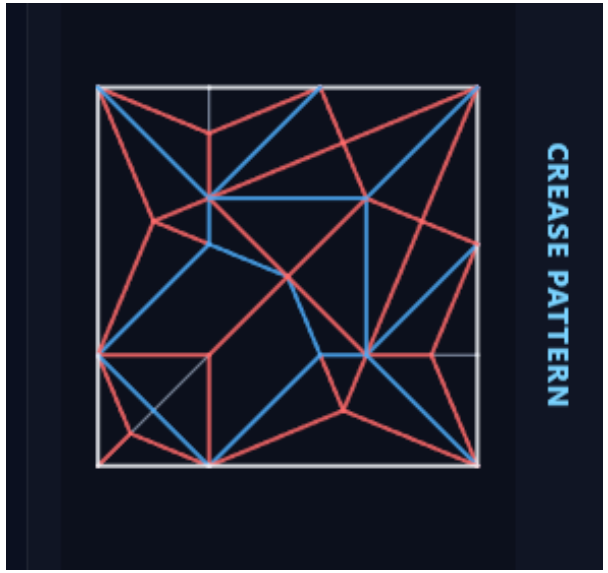
Step 4: simulate folded form

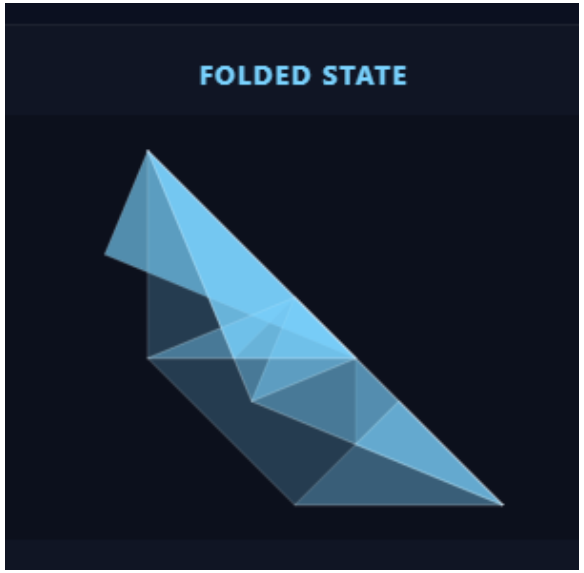




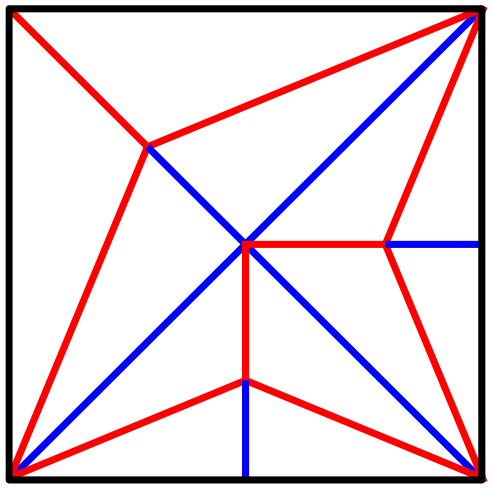
This process is a well known algorithm in origami

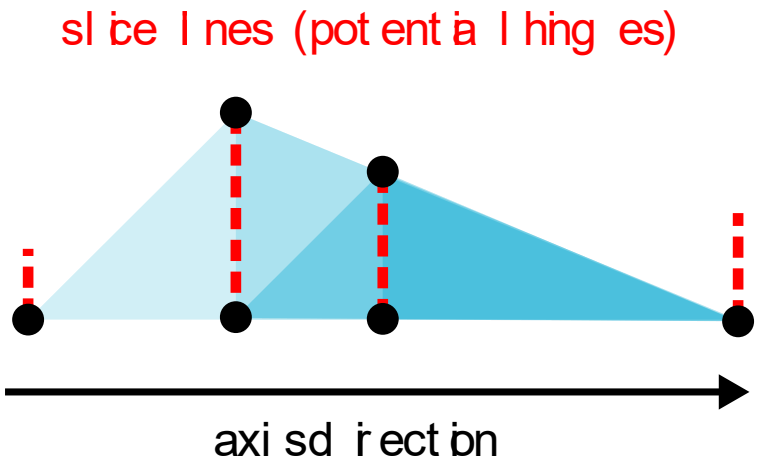
Step 4: extract tree structure

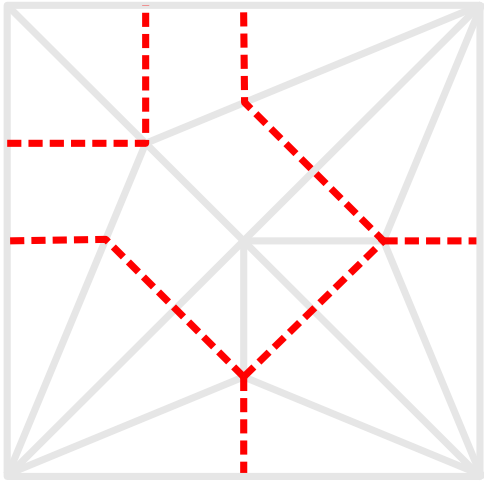


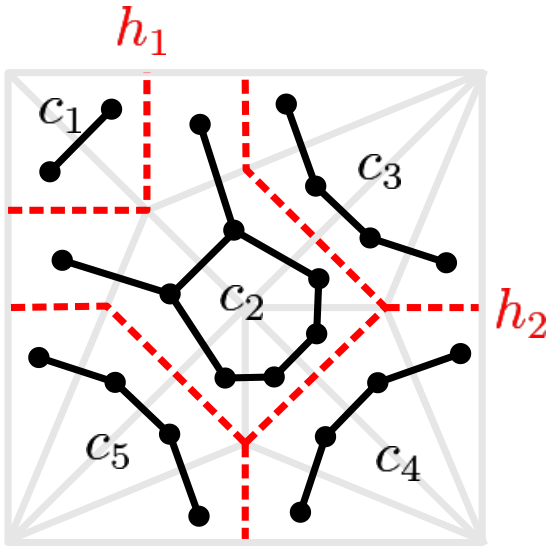


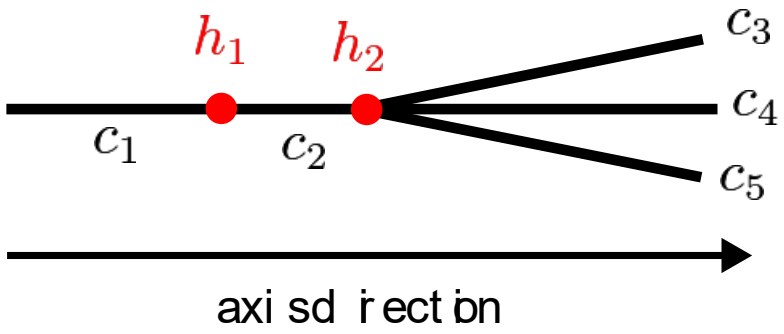
Tree extraction algorithm



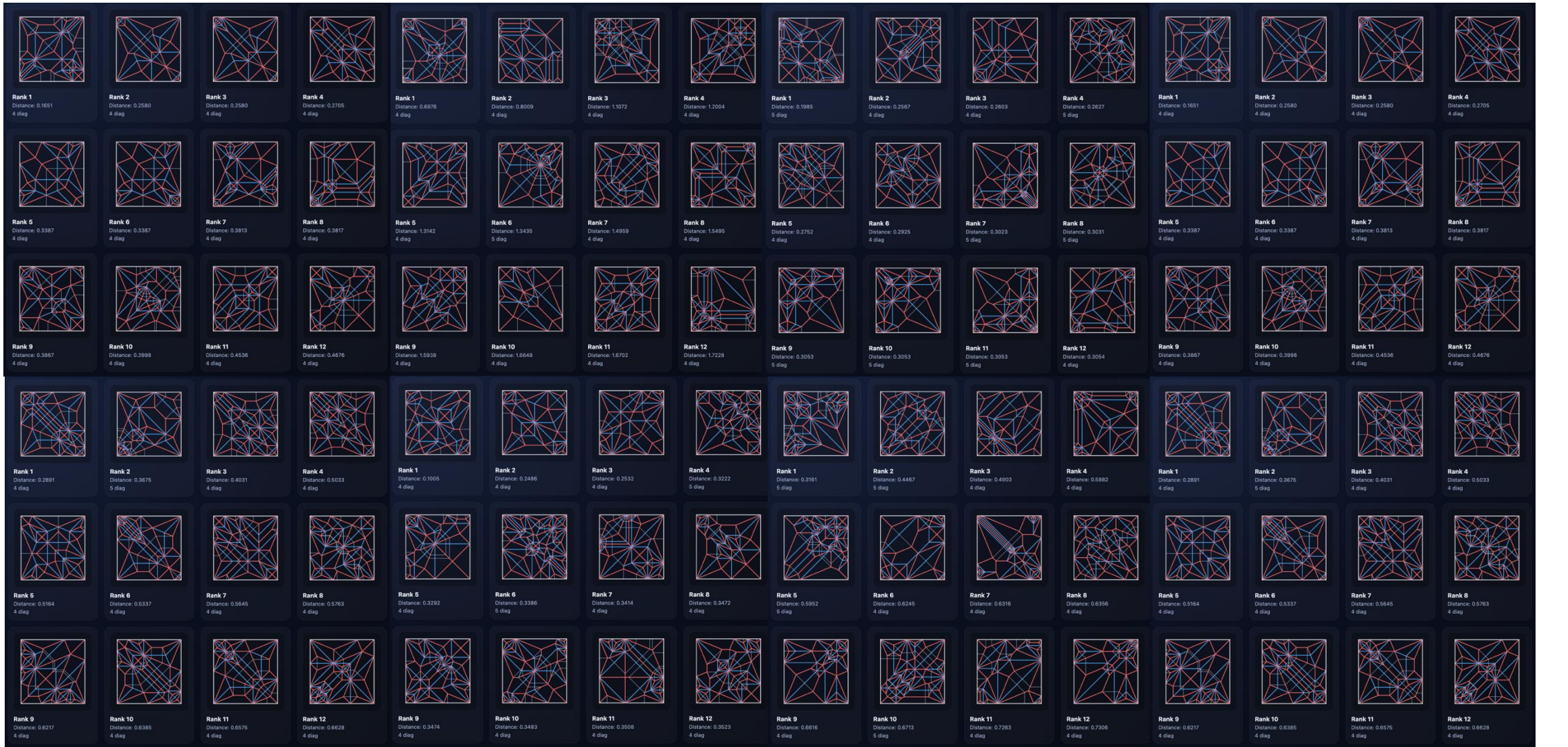






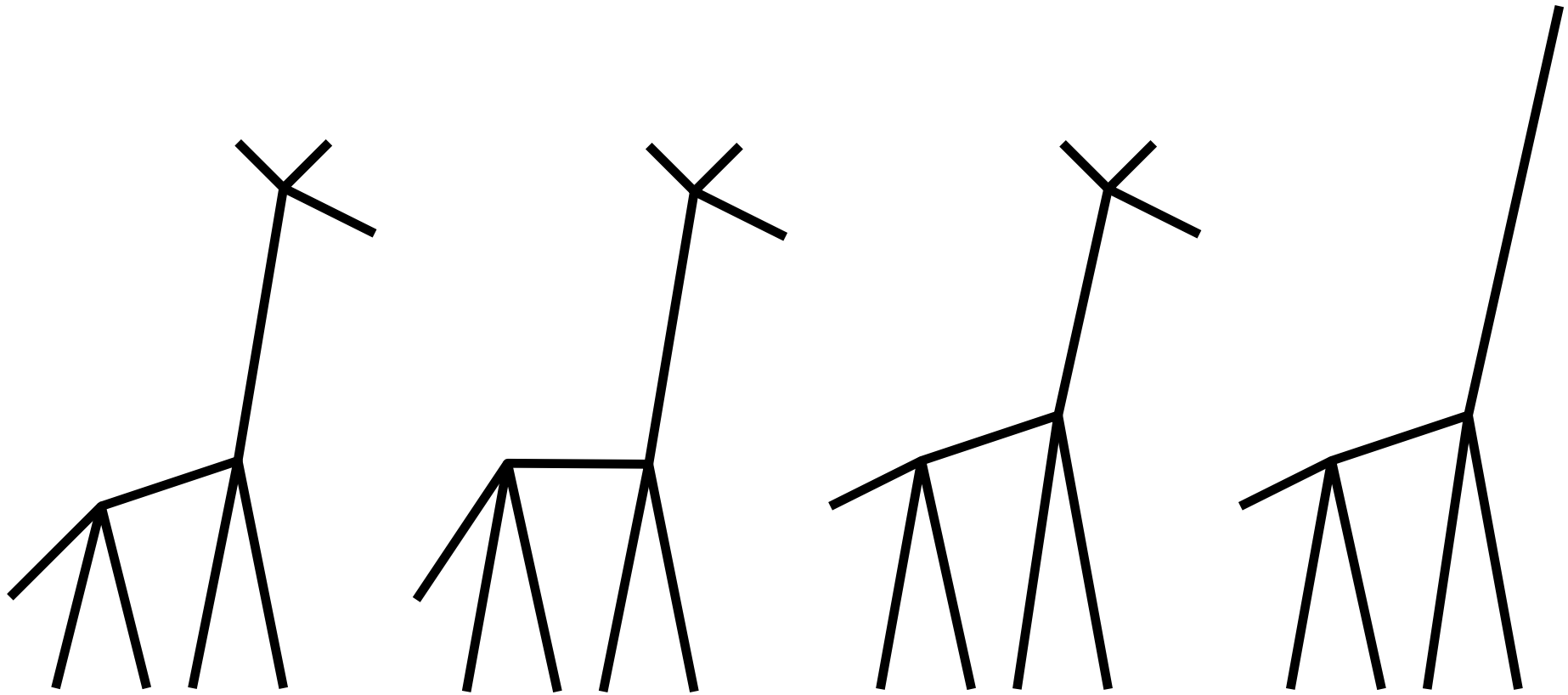


Result: millions of crease patterns and their trees



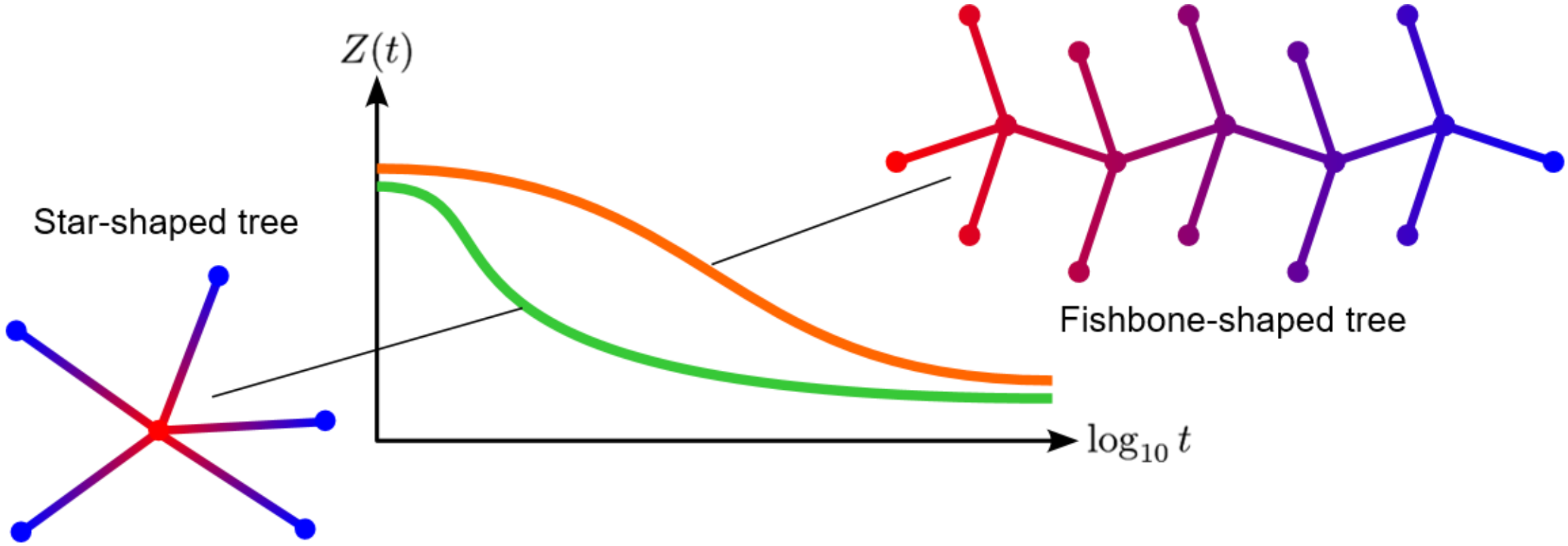
Part 2: Database Lookup

How to tell if trees are similar?

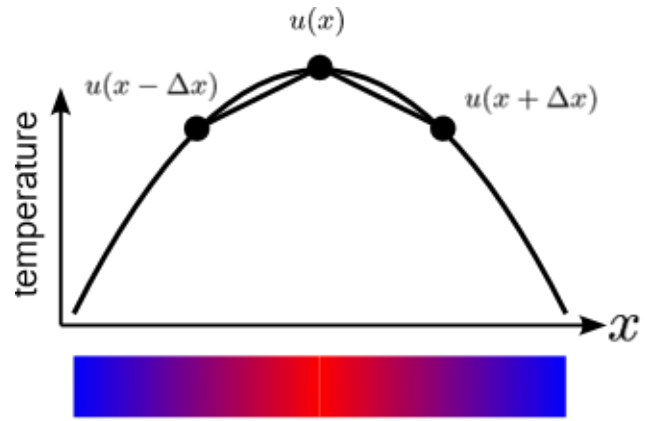


Key idea

Trees with different shapes will cool down at different rates

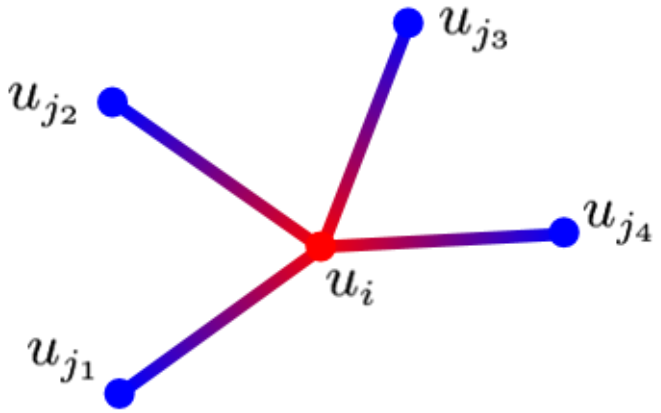


The heat equation (from physics)



$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

1d continuous heat equation



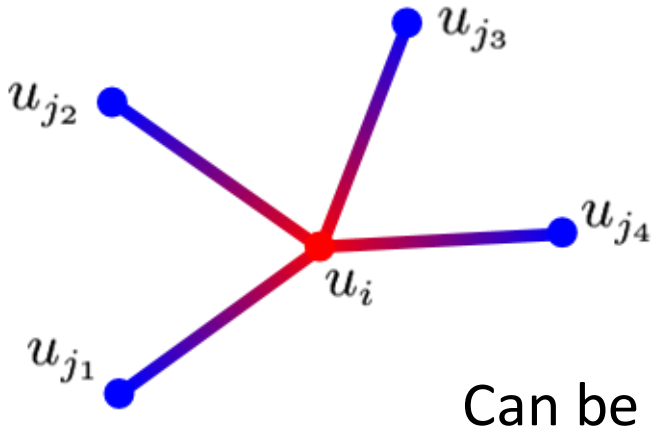
$$\frac{du_i}{dt} = \sum_{j \sim i} \frac{u_j - u_i}{L_{ij}} \quad L = D - A \quad \frac{d\vec{u}}{dt} = -L\vec{u}$$

Discrete graph heat equation

The temperature change at a point in a material depends on the temperature of its neighbors. If it's hotter than its neighbors, it cools down.

The heat kernel trace

$$\frac{d\vec{u}}{dt} = -L\vec{u} \quad \Rightarrow \quad \vec{u}(t) = e^{-tL}\vec{u}(0) \quad \Rightarrow \quad \vec{u}(t) = H(t)\vec{u}(0)$$



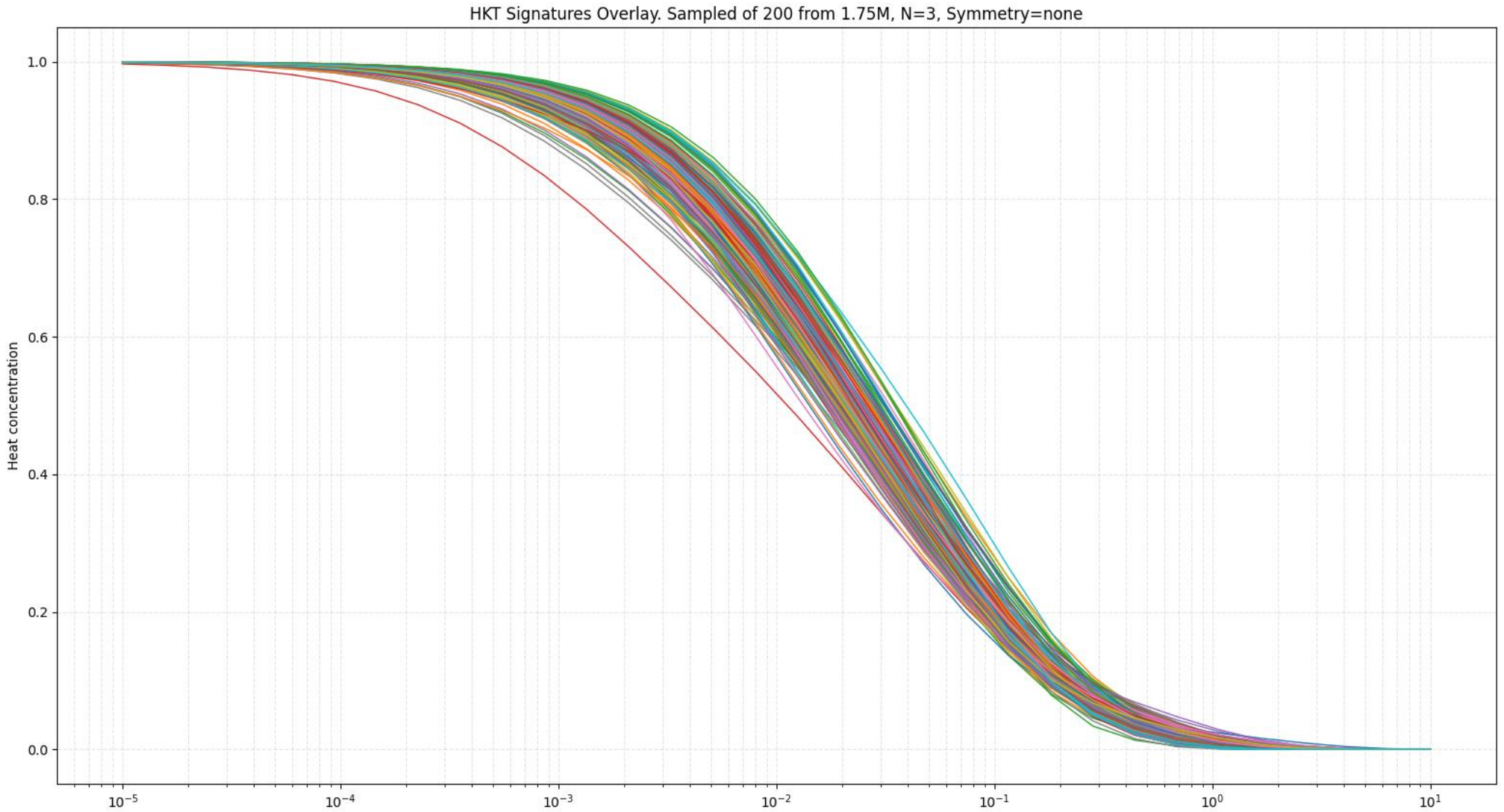
Define: $H(t) = e^{-tL}$

Define: $Z(t) = \mathrm{tr}(H(t)) = \sum_{i=1}^{N} H_{ii}(t)$

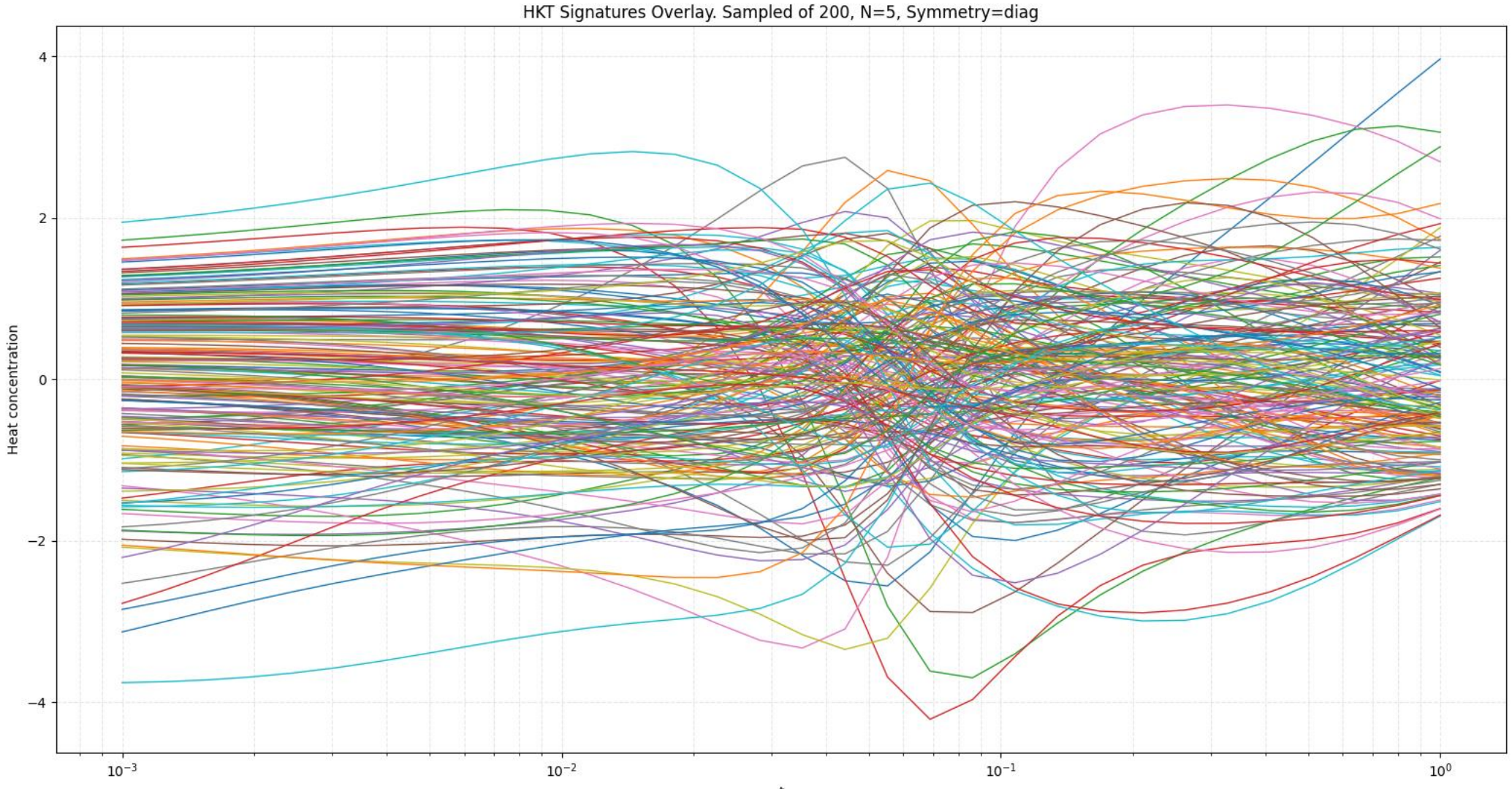
Can be computed as: $Z(t) = \sum_{i=1}^{N} e^{-t\lambda_i}$

We can fully describe the heat distribution of the tree over time

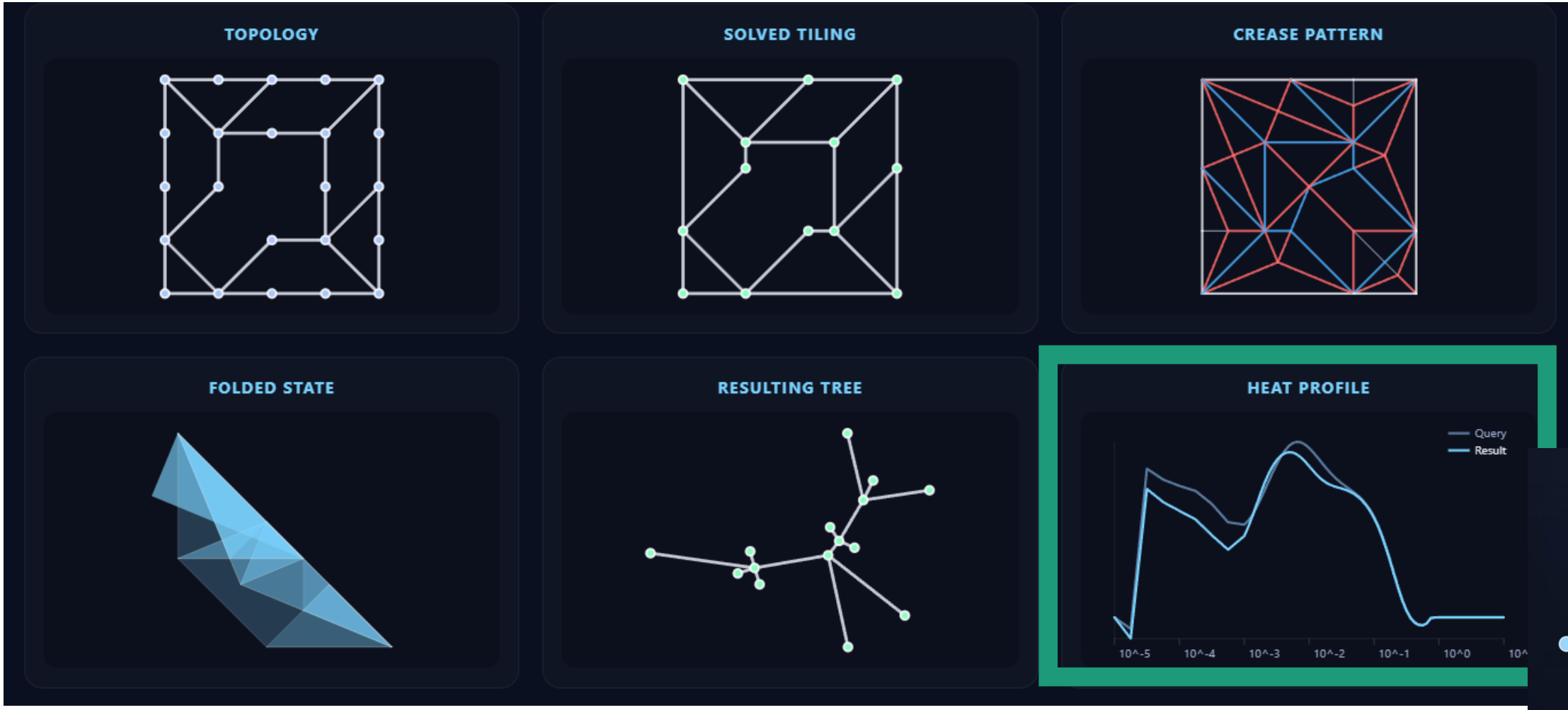
All the trees follow roughly the same curve

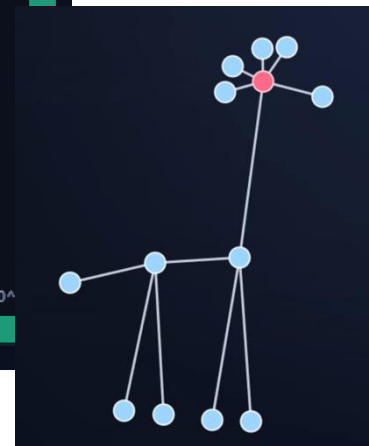


Use statistics to amplify the differences

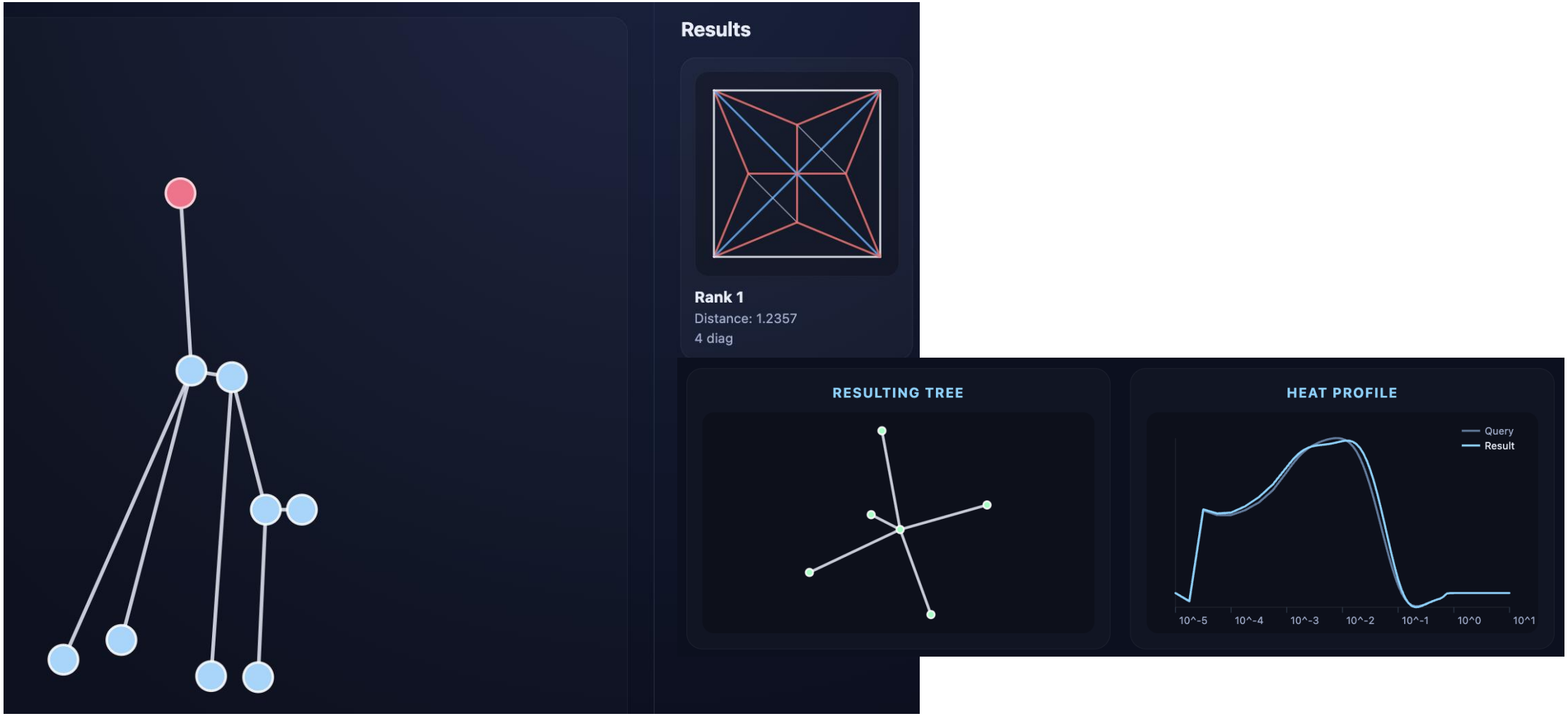


Compare the input curve to every curve in the database

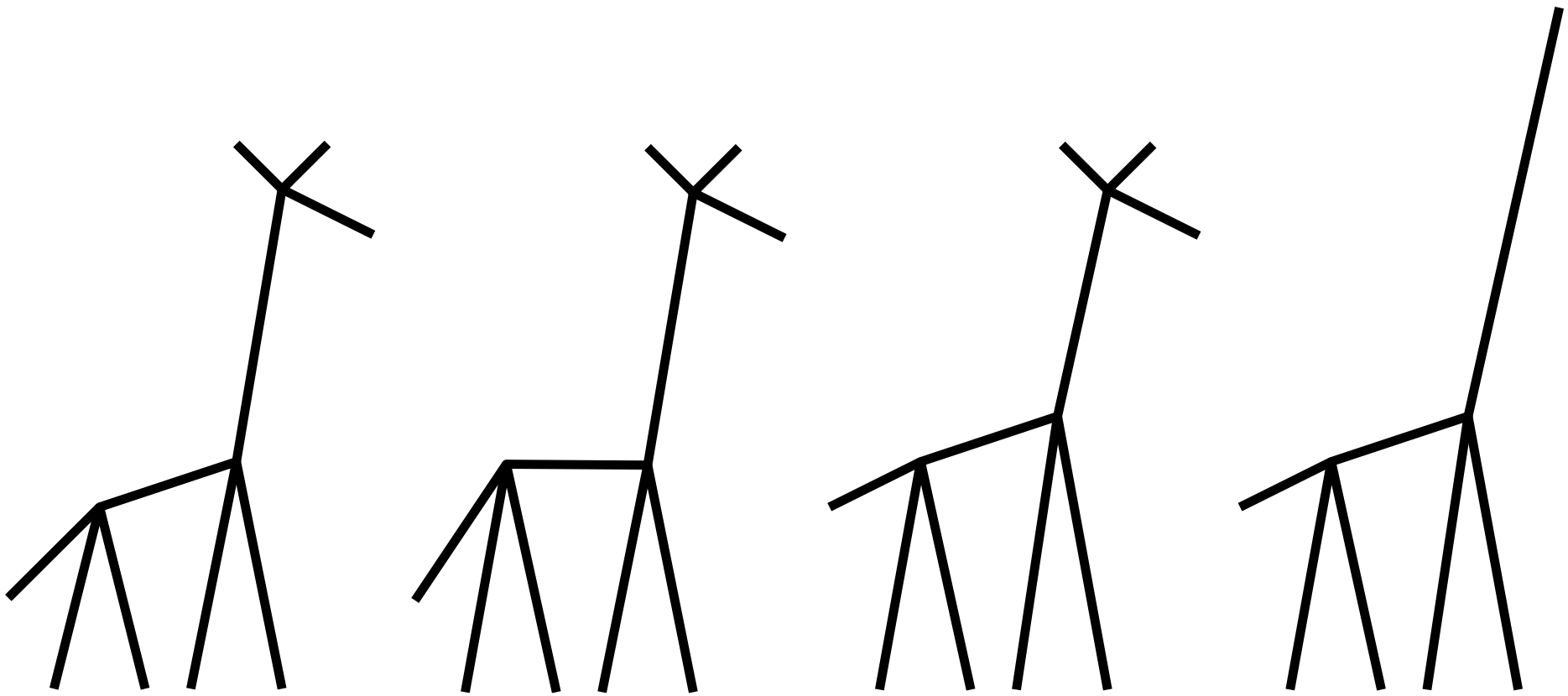




Stubs/distortions will not disrupt the heat diffusion



Stubs/distortions will not disrupt the heat diffusion

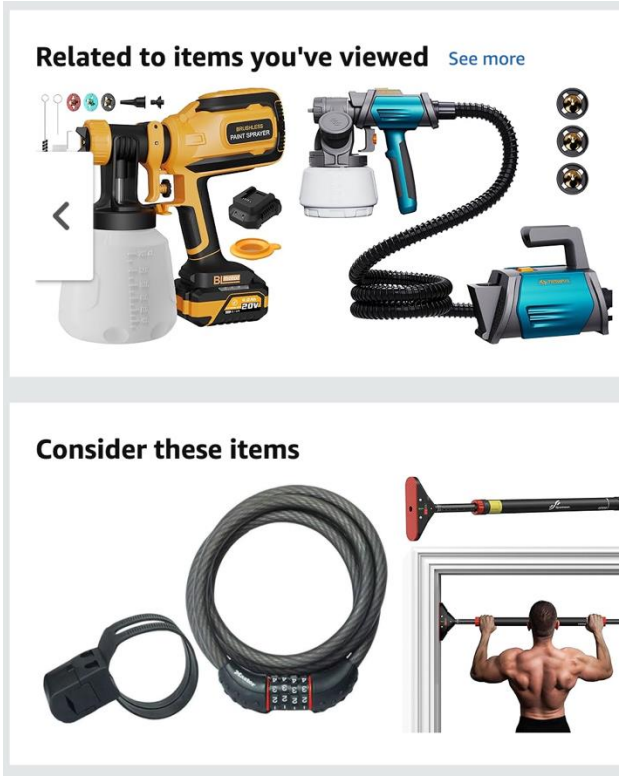


Approximations are ok

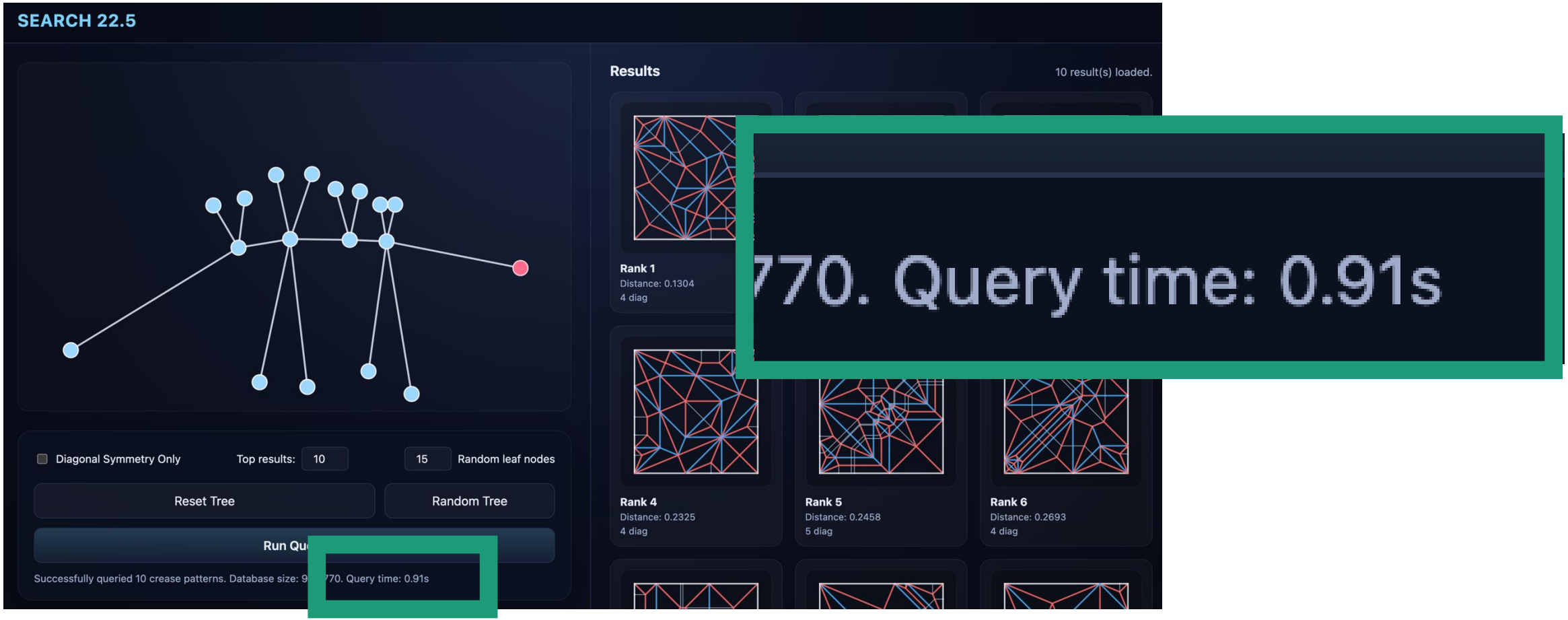
Use FAISS to quickly look through database

FAISS: Facebook's public code library to quickly look through large databases for matches

amazon



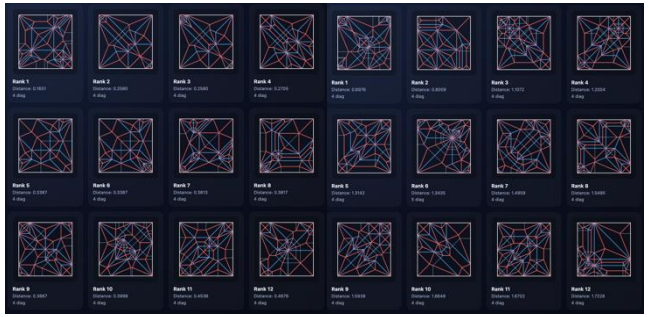
That’s why this runs so quickly

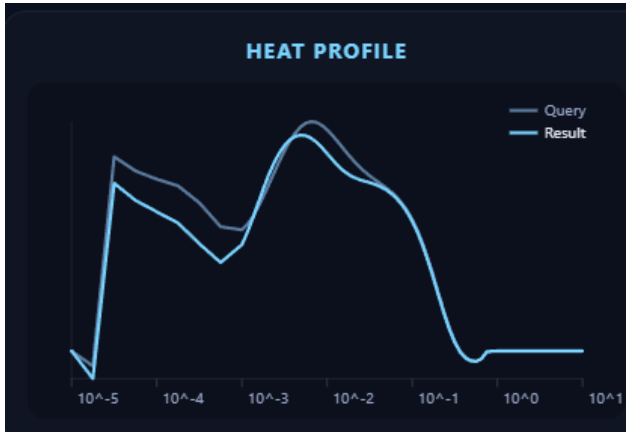


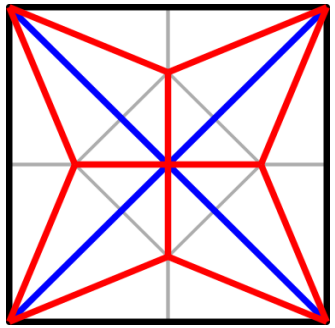
What's with the name?

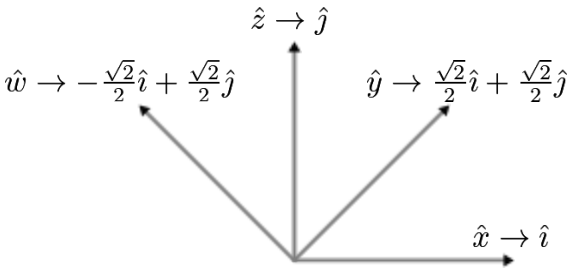
- Spectral
- Embedding
- Archive for
- Rational
- Crease pattern
- Hyperspace

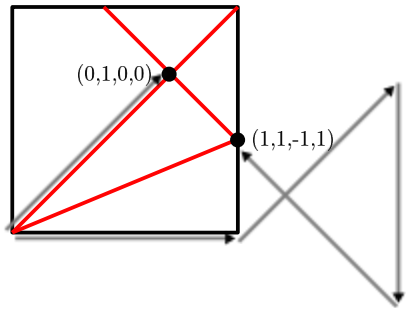
$$Z(t) = \sum_{i=1}^{N} e^{-t\lambda_i}$$











Part 3: Discussion

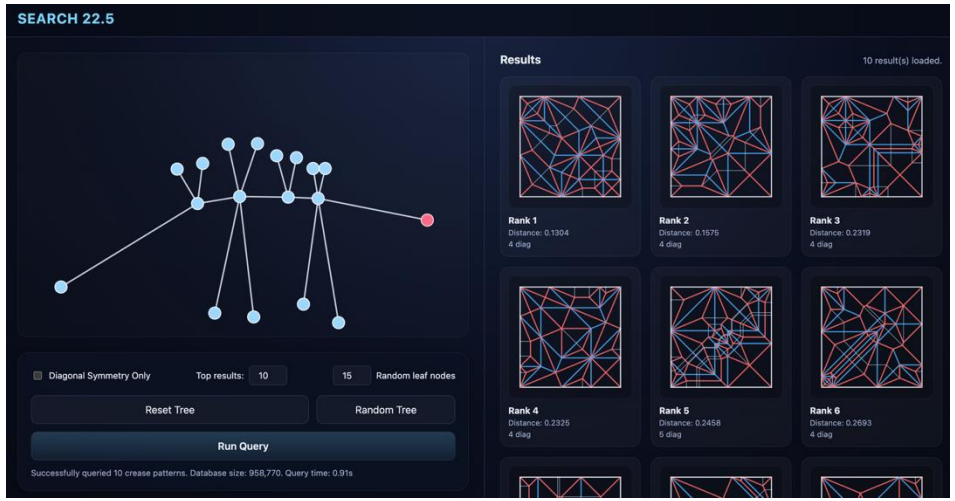
Current limitations

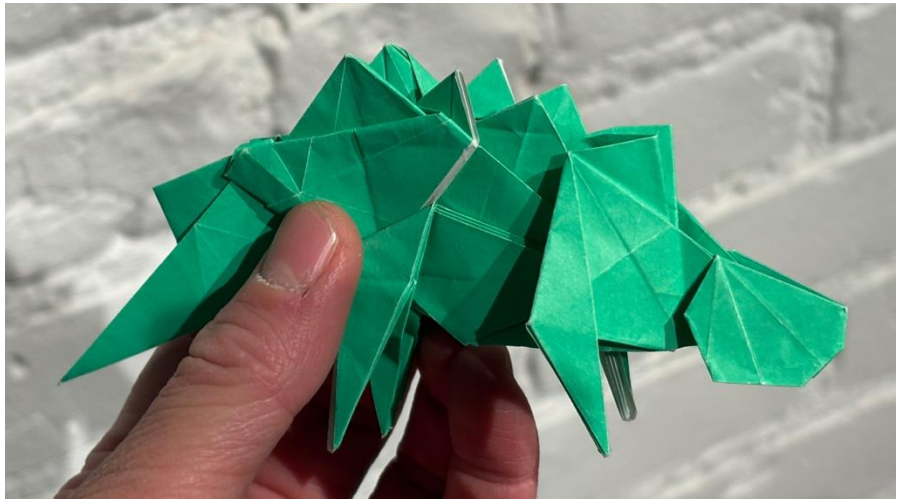
- Design space is infinite—1 million cps is still nothing
  - Hoping for the perfect cp is like monkeys on the typewriter
  - Often the closest tree is not close enough
- Cannot control specific flap locations or symmetry

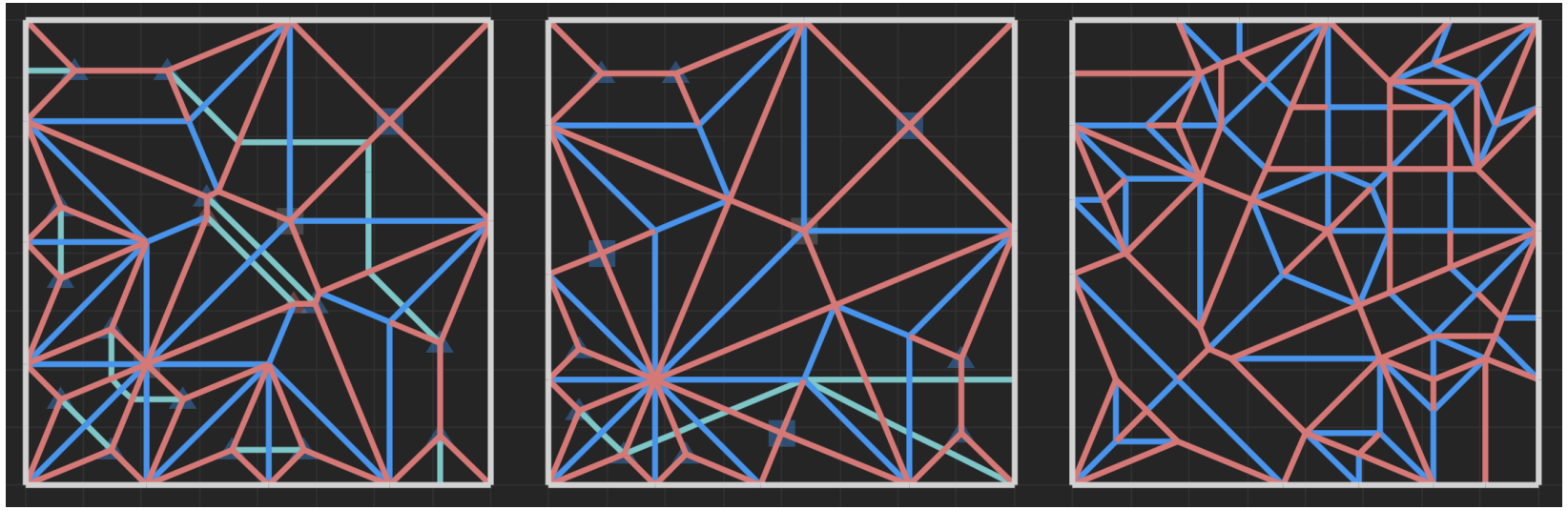
Not at all a replacement for human intuition!

- Initial tree itself is not trivial
- If the output tree is different, will need to shape/point split
- The “best” result might not be the top rank

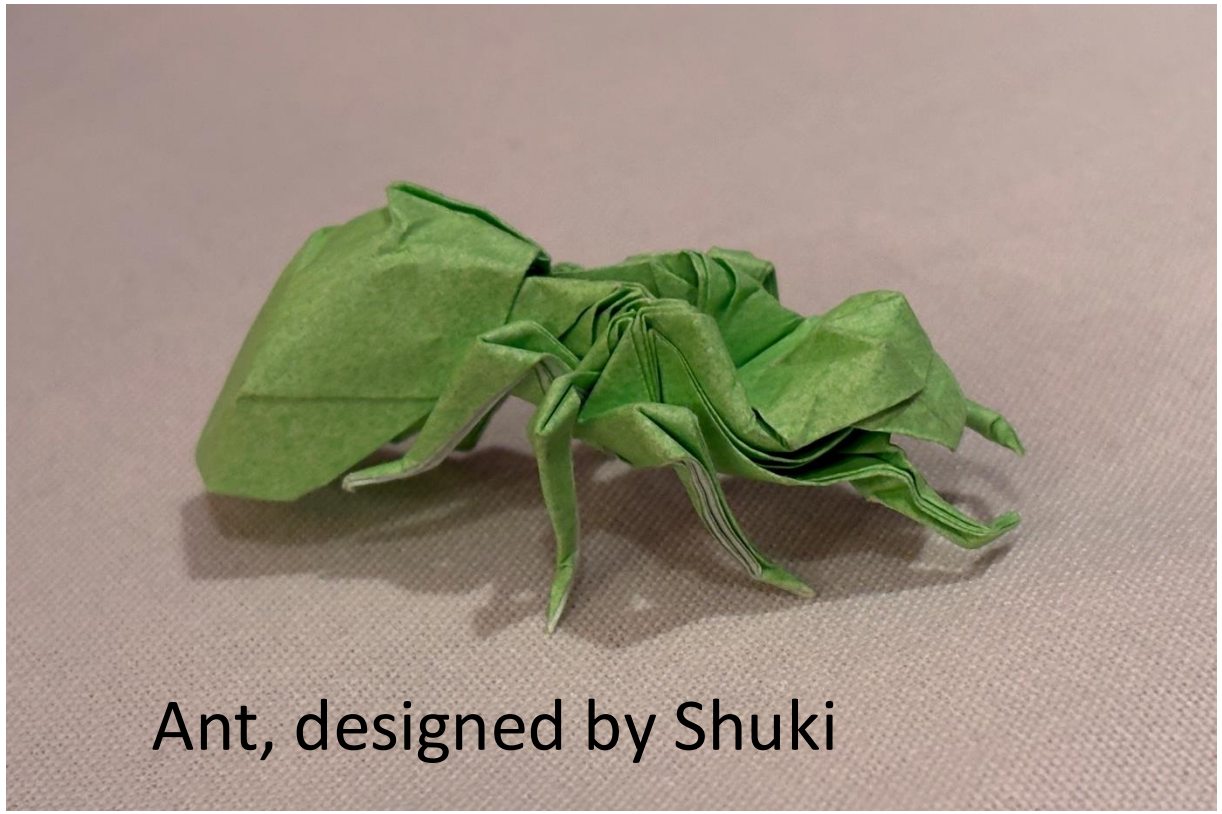
Current Utility: ~50% of the design process







Who gets the credit for the design?



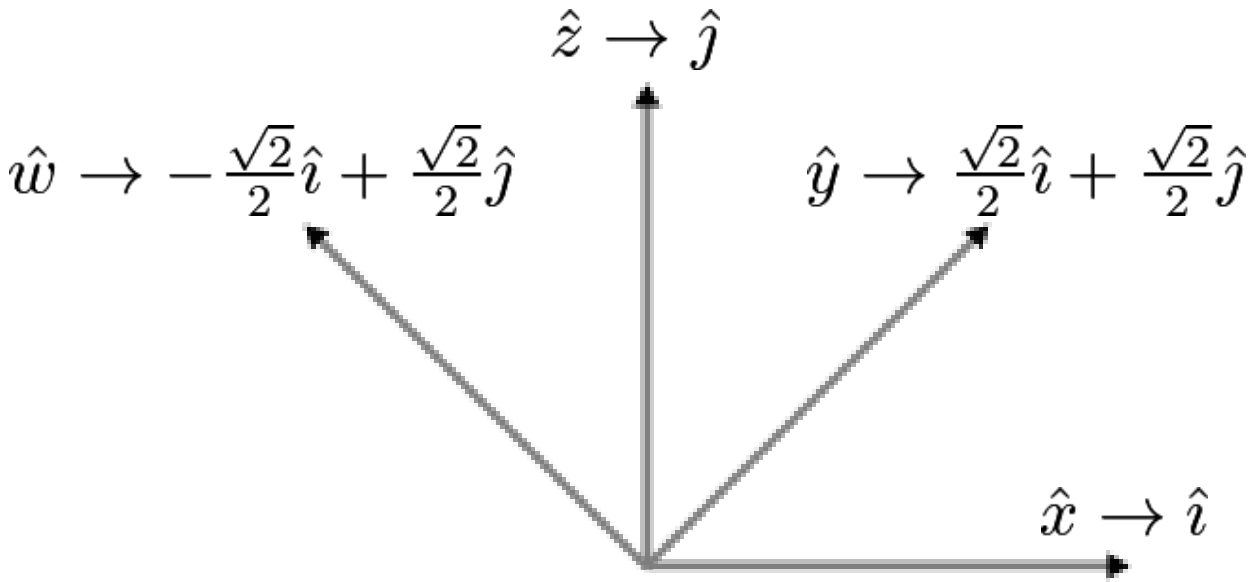
Thank you!
Questions / Open Discussion

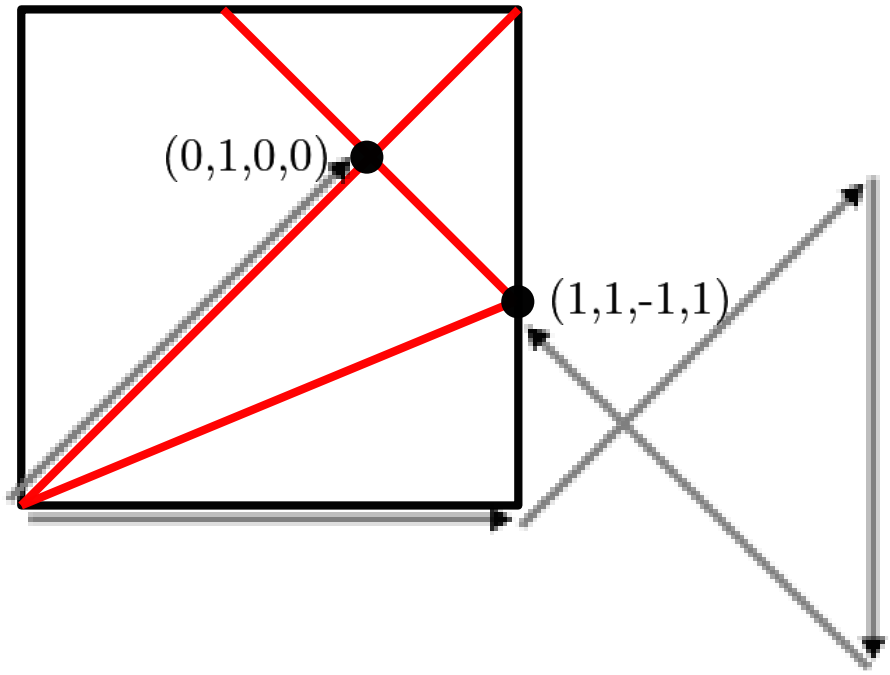
Is this the future of 22.5 design?

Ann Arbor, Michigan, USA CFC 6 May 14-17, 2026

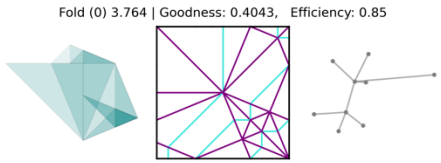
Appendix

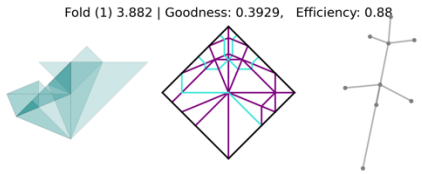
4D basis

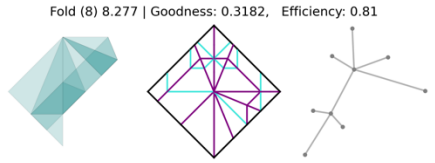

$\hat{z} \to \hat{j}$
$\hat{w} \to -\frac{\sqrt{2}}{2}\hat{\imath} + \frac{\sqrt{2}}{2}\hat{\jmath}$
$\hat{y} \to \frac{\sqrt{2}}{2}\hat{\imath} + \frac{\sqrt{2}}{2}\hat{\jmath}$
$\hat{x} \to \hat{\imath}$

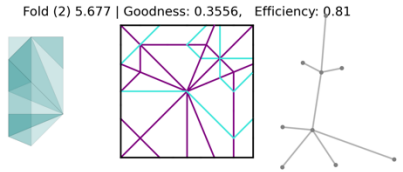

(0,1,0,0)
(1,1,-1,1)

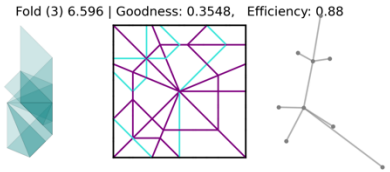
Non-uniaxial 22.5?

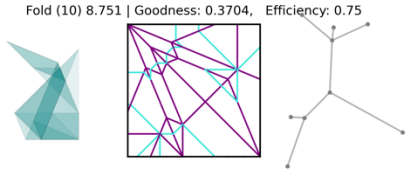


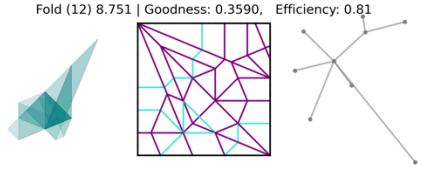


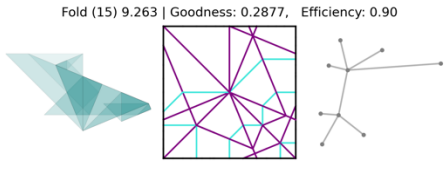




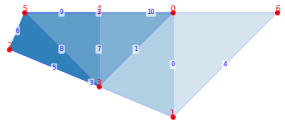




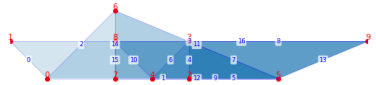




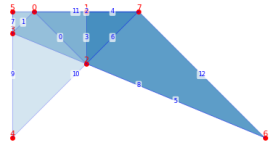
Fold 0



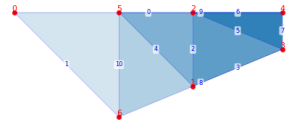
Fold 1



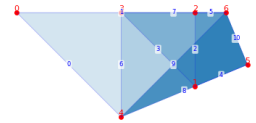
Fold 2



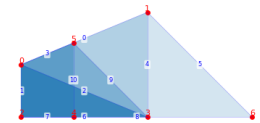
Fold 3



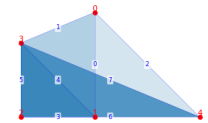
Fold 4



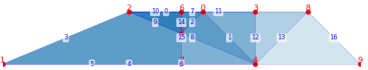
Fold 5



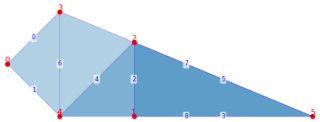
Fold 6



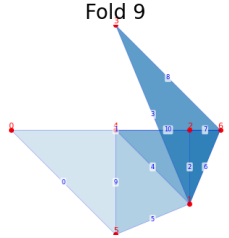
Fold 7



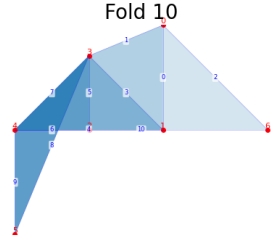
Fold 8



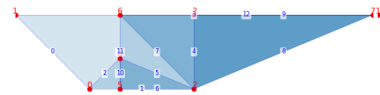
Fold 9



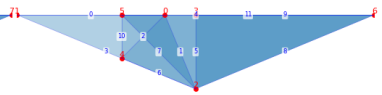
Fold 10



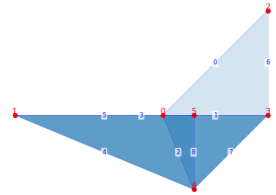
Fold 11



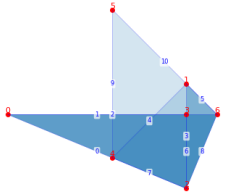
Fold 12



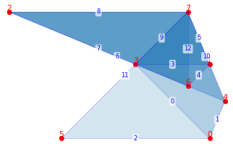
Fold 13



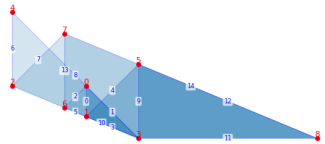
Fold 14



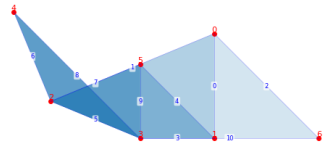
Fold 15



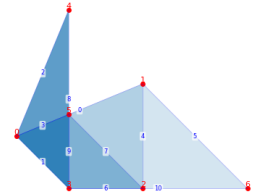
Fold 16



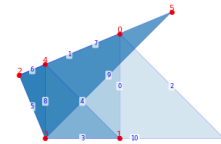
Fold 17



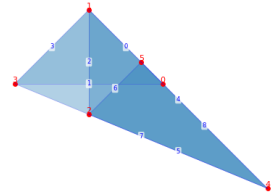
Fold 18



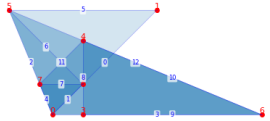
Fold 19



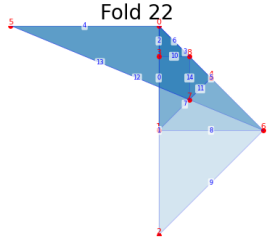
Fold 20



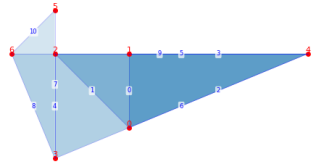
Fold 21



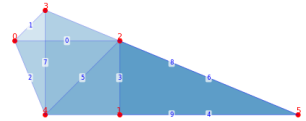
Fold 22



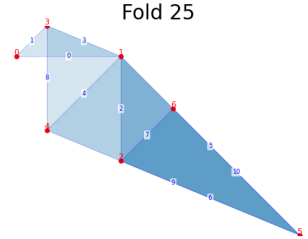
Fold 23



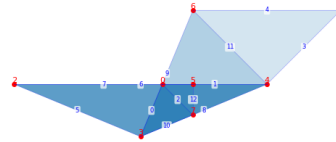
Fold 24



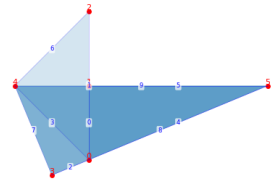
Fold 25



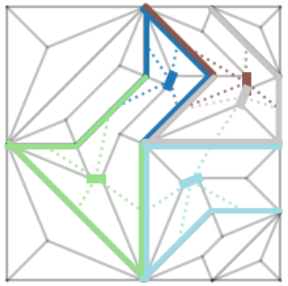
Fold 26



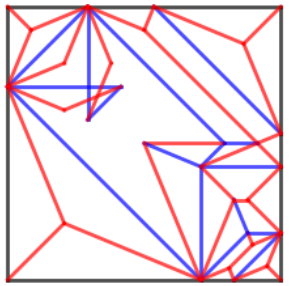
Fold 27



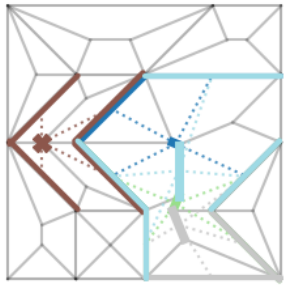
Graph 1: Before



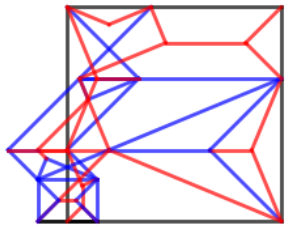
Graph 1: After



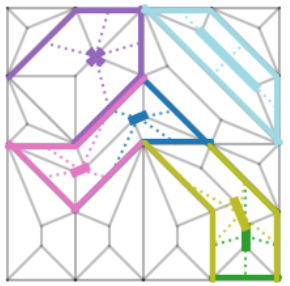
Graph 2: Before



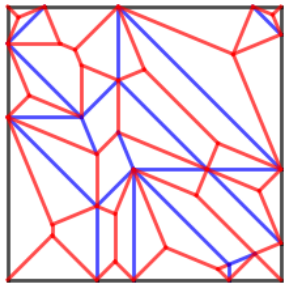
Graph 2: After



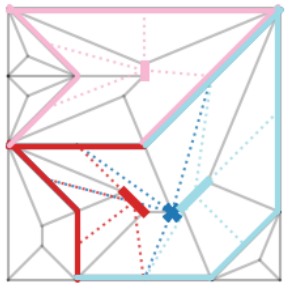
Graph 3: Before



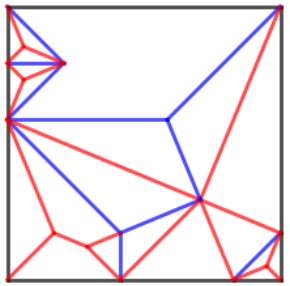
Graph 3: After



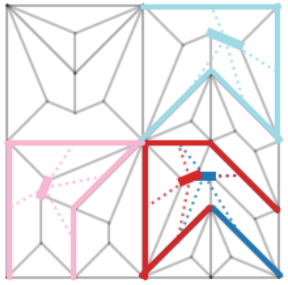
Graph 4: Before



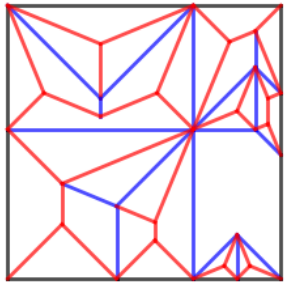
Graph 4: After



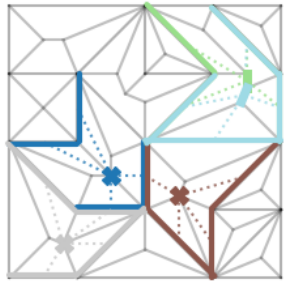
Graph 5: Before



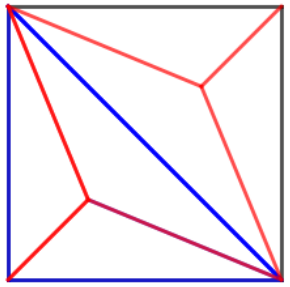
Graph 5: After



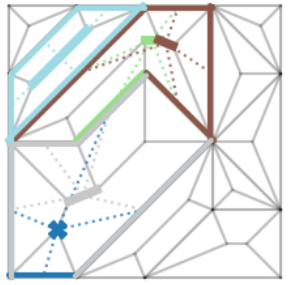
Graph 6: Before



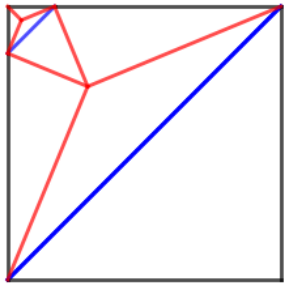
Graph 6: After



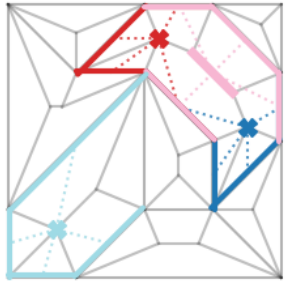
Graph 7: Before



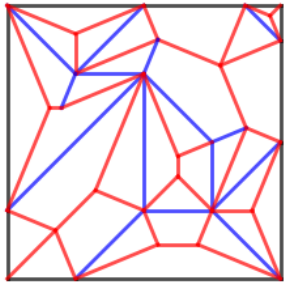
Graph 7: After



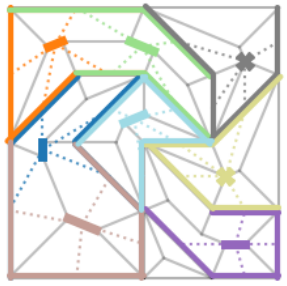
Graph 8: Before



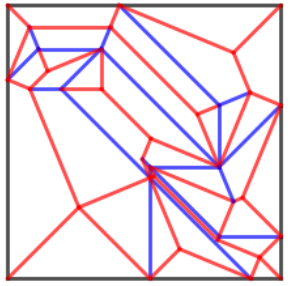
Graph 8: After



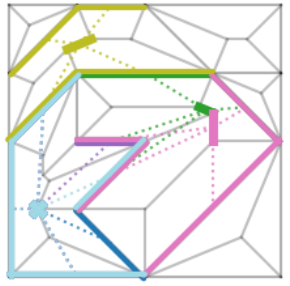
Graph 9: Before



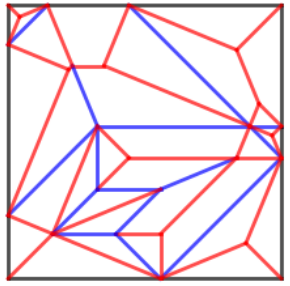
Graph 9: After



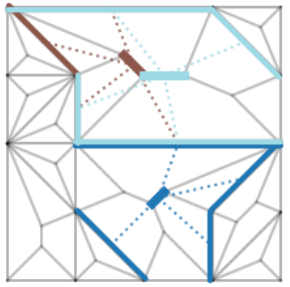
Graph 10: Before



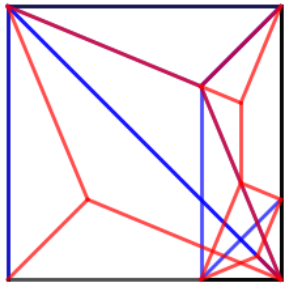
Graph 10: After



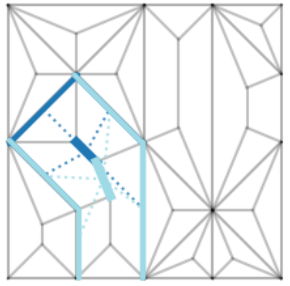
Graph 11: Before



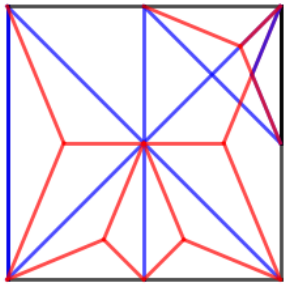
Graph 11: After

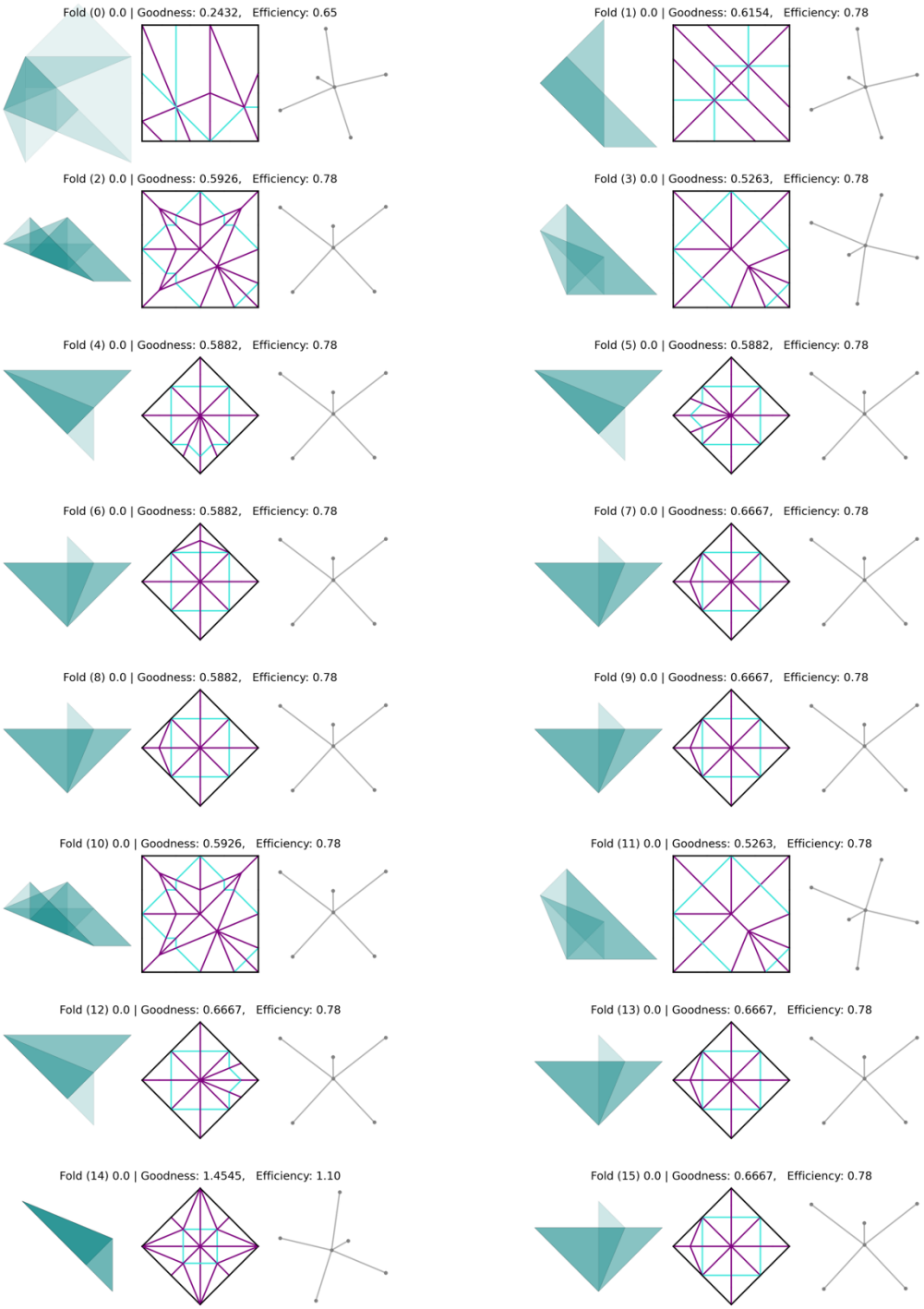


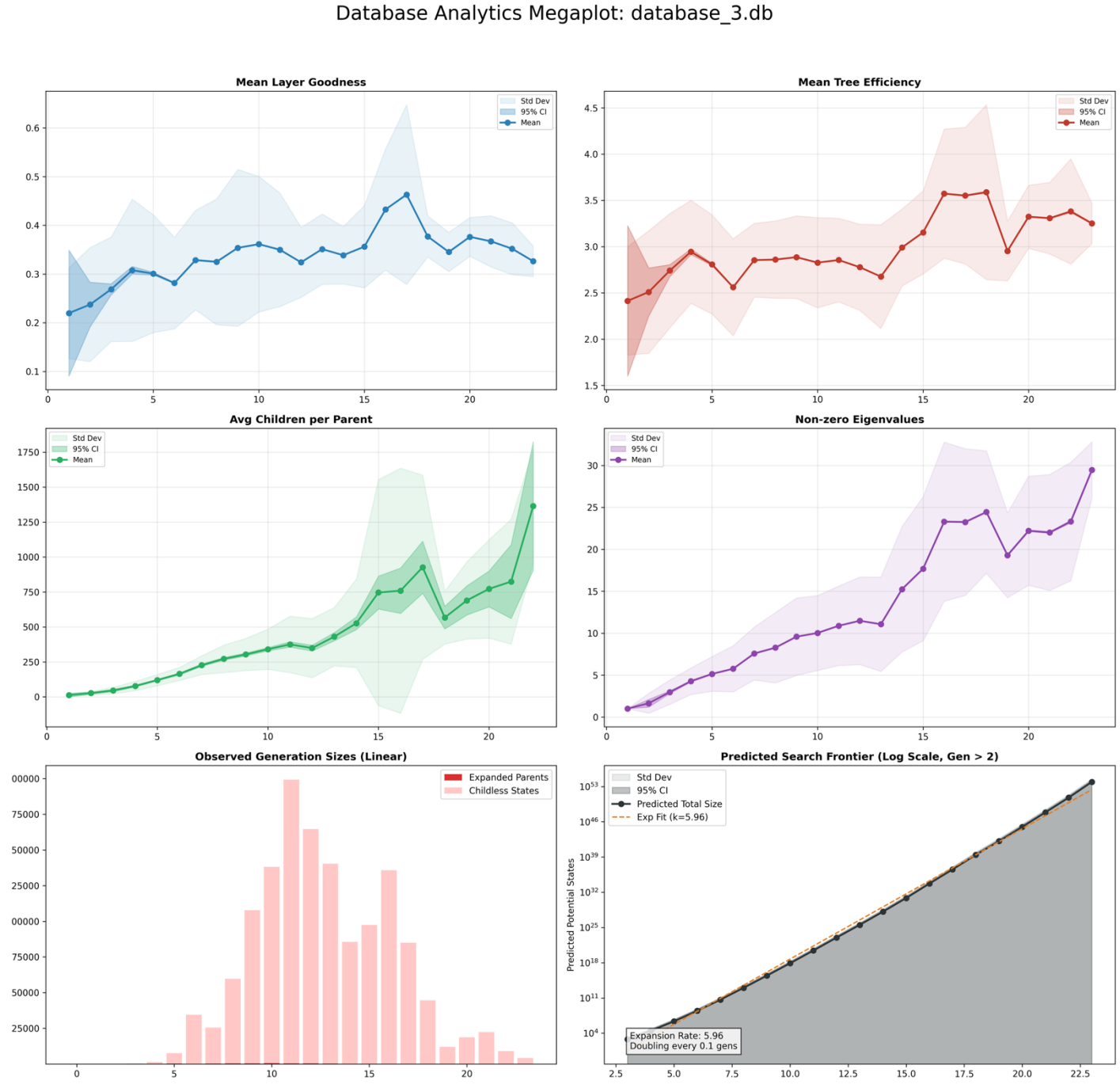
Graph 12: Before



Graph 12: After




Fold (0) 0.0 | Goodness: 0.2432, Efficiency: 0.65
Fold (1) 0.0 | Goodness: 0.6154, Efficiency: 0.78
Fold (2) 0.0 | Goodness: 0.5926, Efficiency: 0.78
Fold (3) 0.0 | Goodness: 0.5263, Efficiency: 0.78
Fold (4) 0.0 | Goodness: 0.5882, Efficiency: 0.78
Fold (5) 0.0 | Goodness: 0.5882, Efficiency: 0.78
Fold (6) 0.0 | Goodness: 0.5882, Efficiency: 0.78
Fold (7) 0.0 | Goodness: 0.6667, Efficiency: 0.78
Fold (8) 0.0 | Goodness: 0.5882, Efficiency: 0.78
Fold (9) 0.0 | Goodness: 0.6667, Efficiency: 0.78
Fold (10) 0.0 | Goodness: 0.5926, Efficiency: 0.78
Fold (11) 0.0 | Goodness: 0.5263, Efficiency: 0.78
Fold (12) 0.0 | Goodness: 0.6667, Efficiency: 0.78
Fold (13) 0.0 | Goodness: 0.6667, Efficiency: 0.78
Fold (14) 0.0 | Goodness: 1.4545, Efficiency: 1.10
Fold (15) 0.0 | Goodness: 0.6667, Efficiency: 0.78


Database Analytics Megaplot: database_3.db
Mean Layer Goodness
Std Dev
95% CI
Mean
Mean Tree Efficiency
Avg Children per Parent
Non-zero Eigenvalues
Observed Generation Sizes (Linear)
Expanded Parents
Childless States
Predicted Search Frontier (Log Scale, Gen > 2)
Predicted Total Size
Exp Fit (k=5.96)
Predicted Potential States
Expansion Rate: 5.96
Doubling every 0.1 gens