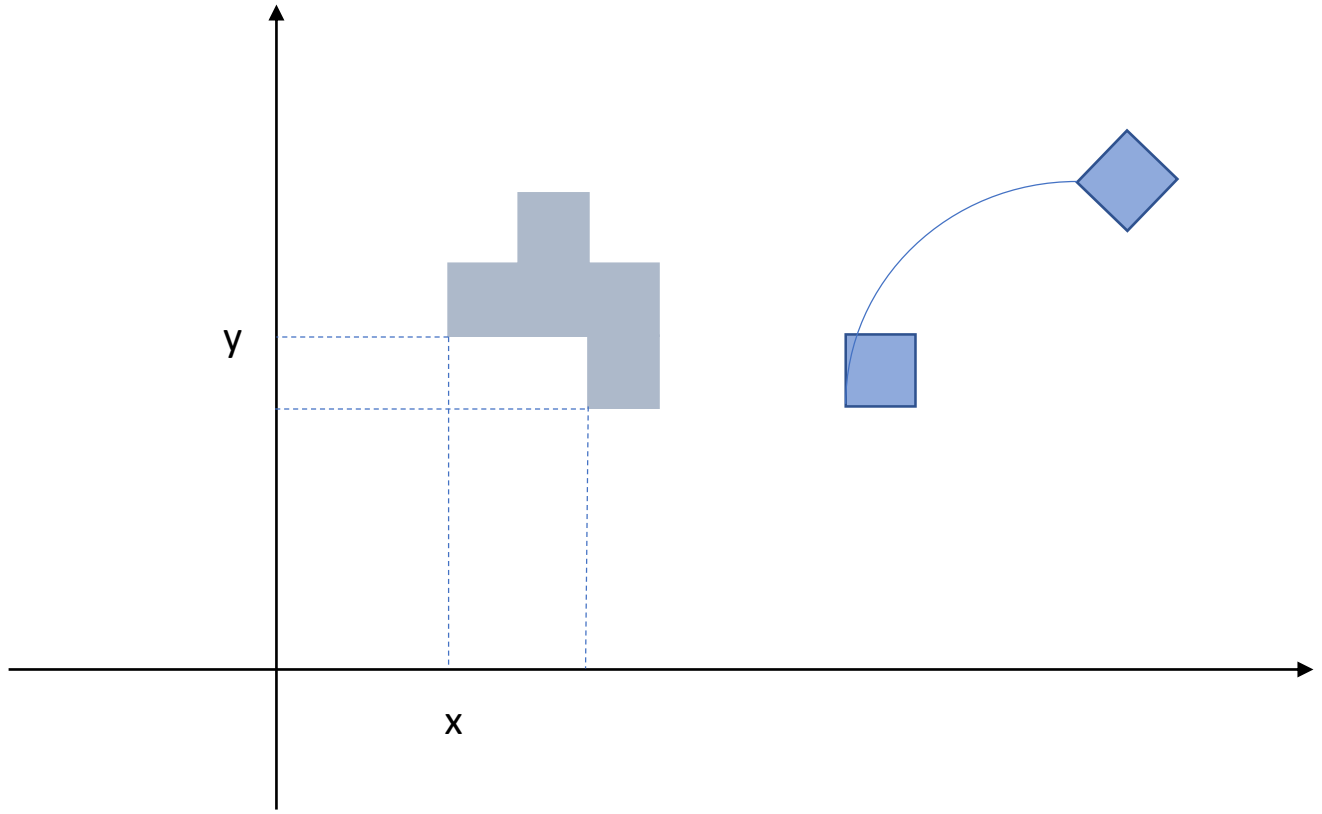
Polyomino tilings and exact cover

Houssam El Cheairi

Abdelhafid Souilmi

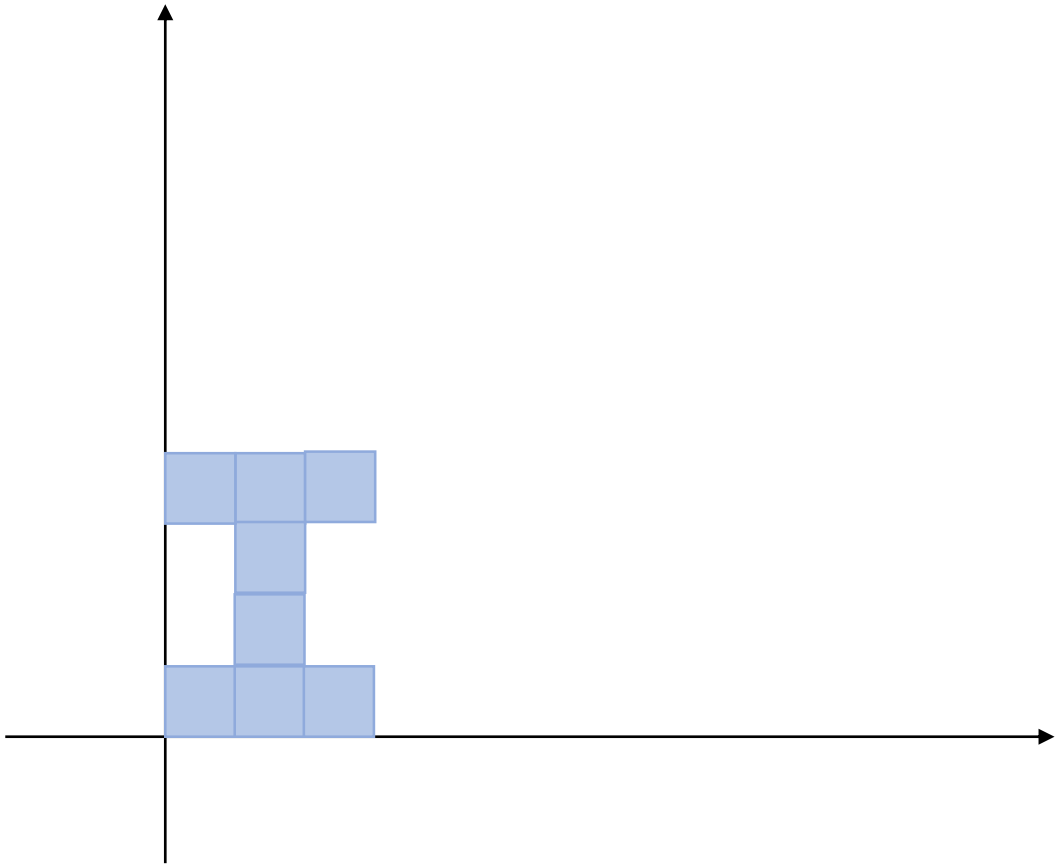
Polyomino representation :



Representing a **polyomino** using the squares it's made of.
Each **square** is represented by the left lower point.

We define transformations on squares and through them to polyominoes.

Cannonical Representation of a Polyomino :



Lowest square on the left side at the origin of the plan.

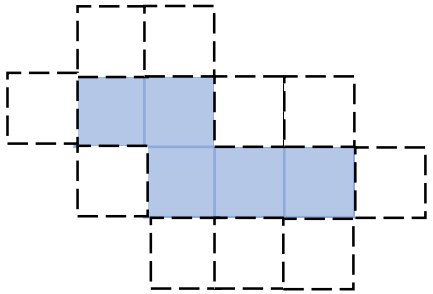
Check in O(n) if two polyominoes are fixed or free.

Generating Polyominoes :

Fixed & Free:

We will generate fixed polyominoes using induction : having a polyomino of **size n**, we can **increase** its area to n+1 b adding squares, then **remove fixed or free duplicates** using the canonical representation. This is the most costly operation.

Creating polyominoes this way has an exponential complexity.



Using an area 5 polyomino to create an area 6 polyomino,
with dashed squares being potentiel add-ons.

Generating polyominoes using Redemeiler's Algorithm:

- Using this Algorithm we will remove the ''Check for duplicates'' step.
- We identify the polyomino using the canonical representation and using a sorted list (x first then y if x coordinates equal) of the squares representing it.

For *fixed polyominoes*, we use a **depth-first traversal Tree**:
Each child polyomino in the tree consists of its parent plus one new adjacent cell. The cell is chosen so that no older brother or ancestor's older brother contains it. Hence at the depth n, we get all fixed polyominoes with an area n.

For *free polyominoes,* we generate the Orbit (result of all Isometries to generate the free polyominoes)and then compare them.

Orbit of a polyomino of size 3: (with all rotations)

Initial Polyomino

Orbit

Exact cover problem :

Using the backtracking method, and tried to improve it using the heuristic however, it didn’t improve the outcome for :

$X=\{1,2,\ldots,n\}$, $C=P(X)$; $X=\{1,2,\ldots,n\}$, $C=\{U \subset X, |U|=2\}$

It hasn’t been able to do so, since for every x the number of subsets x is in, is the same. So choosing the smallest covered element wouldn’t have an effect on our implementation in this case.

Dancing Links :

Classes

Data

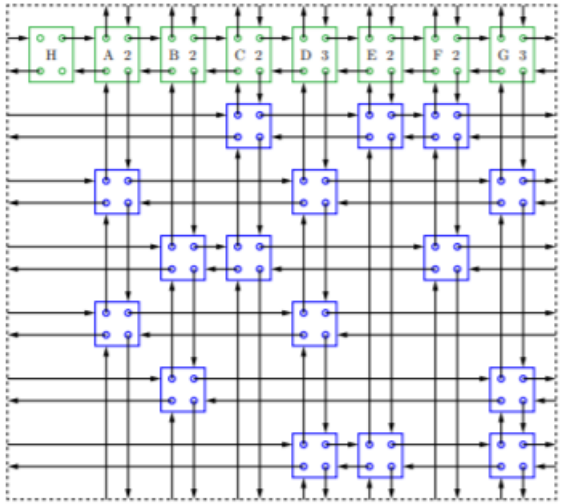
- Upward arrow
- Downward arrow
- Rightward arrow
- Leftward arrow
- Column
- Set

Column

- Number of 1’s in the column
- Name of the column

Dancing Links

- The H Column



From polyomino tilings to exact cover :

-Possible repetitions :

To convert a GroundSet G and a Tiles set T, we use a hash function for each square :

$$Hash(Point(x, y)) = n.(y + \frac{n}{2}) + x + \frac{n}{2}$$

With n being a Key sufficently big, we can get x and y directly from the euclidean division : *Each square is represented by an integer.*

For each polyomino, we generate all translated polyominoes that fit inside G, for each one of these translated polyominoes we get a subset in G, we change every square to an integer using the Hash function.

Now we have an integer set X reprensenting G and a set of subsets C representing all polyominoes (with their translations).

-No repetition :

We implement this constraint in the dancing links structure by adding fictional columns.

For each polyomino, we define $\mathcal{A}_j$ as the set with all polyominoes resulting from the translations (and rotations if free) that are in G.

For each $\mathcal{A}_j$,we create a fictional column, and if a polyomino in C is in $\mathcal{A}_j$ we put a 1 in the added column.

Since for each the sum of each column needs to be 1, at the end, we are sure to use every polyomino exaclty once

Tilings of some polyominoes using free pentaminoes :

For $\mathcal{F}_a$,we found : 404 possible tilings,
For $\mathcal{F}_b$,we found : 374 possible tilings,
For $\mathcal{F}_c$,we found : 0 possible tilings.

$\mathcal{F}_a$ $\mathcal{F}_b$ $\mathcal{F}_c$

Tiling a 6 x 3 rectangle by k-size fixed polyominoes:

k	Nbr of tilings 2nd case
1	1
2	41
3	170
4	0
5	0
6	132

Tiling a n x n square by k-size fixed polyominoes:

k,n	Nbr of tilings 1st case
1	1
2	2
3	10
4	117
5	4006
6	451206

We tested the last case by testing each octomino and 4-dilate, we found 10 free octominos satisfying the condition.

Extension : Sudoku

- A Sudoku grid needs to satisfy 4 constraints :

  Row-Column, Row-Value, Column-Value and Box-Value

- We create our cover matrix :

  A row for every (Column, Row, Value) triplet

  A column for every (Column, Row), (Column, Value), (Row, Value) (Box, Value) couple

  Every two triplets sharing one of these couples, will have a one on the corresponding column

  Here is an example of how the matrix would look like for 4x4 Sudoku.