Internal Report 2010–4 March 2010

Universiteit Leiden
Opleiding Informatica

On Jigsaw Sudoku Puzzles
and Related Topics

Johan de Ruiter

BACHELORSCRIPTIE

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

On Jigsaw Sudoku Puzzles and Related Topics Bachelor Thesis

Johan de Ruiter, johan.de.ruiter@gmail.com
Leiden Institute of Advanced Computer Science (LIACS)
Leiden University, The Netherlands

March 15, 2010

Abstract

Using memoization and various other optimization techniques, the number of dissections of the $n \times n$ square into n polyominoes of size n is computed for $n \leq 8$. On this task our method outperforms Donald Knuth's Algorithm X with Dancing Links. The number of jigsaw sudoku puzzle solutions is computed for $n \leq 7$. For every jigsaw sudoku puzzle polyomino cover with $n \leq 6$ the size of its smallest critical sets is determined. Furthermore it is shown that for every $n \geq 4$ there exists a polyomino cover that does not allow for any sudoku puzzle solution. We give a closed formula for the number of possible ways to fill the border of an $n \times n$ square with numbers while obeying Latin square constraints. We define a cannibal as a nonempty hyperpolyomino that disconnects its exterior from its interior, where the interior is exactly the size of the hyperpolyomino itself, and we present the smallest found cannibals in two and three dimensions.

Contents

1 Introduction

A *Latin square* of order n is an $n \times n$ grid of numbers of which every row and every column is a permutation of the numbers 1 through n. A *partial Latin square* is a grid of numbers and zero or more blanks, such that it is possible to fill in the blanks with numbers in a way that it results in a Latin square.

A *sudoku puzzle*, as shown in Figure 1, is a partial Latin square of order 9 that can in a unique way be completed into a Latin square obeying the extra requirement that when it is dissected into 9 smaller squares of size 3×3 (as indicated in Figure 1), within each of those 3×3 squares no number is repeated.

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

Figure 1: A homemade sudoku puzzle with digits of Pi [1].

A *polyomino* of size n is defined as a subset of n unit squares of a regular square tiling, such that every unit square in this subset is connected to every other unit square in this subset through a sequence of shared edges [2].

In recent years, solving sudoku puzzles has become a widespread phenomenon. Many newspapers supply their readers with their daily dose, many a bookstore offers several different books and magazines devoted to this puzzle and there is even an annual World Sudoku Championship.

In some variants of mainstream sudoku puzzles, grids are being used with regions other than the regular 3×3 squares. This gives rise to puzzles

where the value of n is no longer necessarily a square. In fact, if we define a *jigsaw sudoku puzzle* to be a partial Latin square of order n, which can in a unique way be completed into a Latin square where a given dissection of the $n \times n$ square into n polyominoes of size n induces extra requirements being that no digit is repeated within the same polyomino, any square grid can now be used and the irregular shapes of the regions give a nice extra twist when solving the puzzles. For an example of a jigsaw sudoku puzzle, see Figure 2.

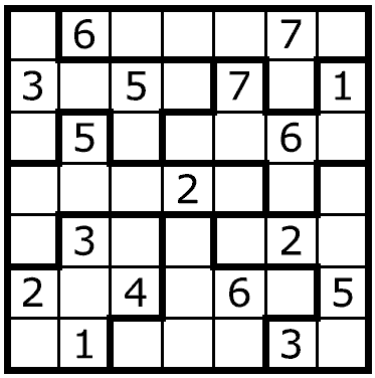

Figure 2: A homemade jigsaw sudoku puzzle for $n = 7$ [3].

This paper first deals with efficiently computing the number of dissections of the $n \times n$ square into n polyominoes of size n (also referred to as *polyomino covers*) for $n \leq 8$, using memoization and various other optimization techniques; see Section 2. Secondly, it describes how the total number of completed jigsaw sudoku puzzles can be computed for $n \leq 7$ and thirdly, for $n \leq 6$ it focuses on *minimal jigsaw sudoku puzzles*, i.e., partial jigsaw sudoku puzzles with the property that there exists no partial jigsaw sudoku puzzle of equal size with less numbers that has a unique solution. The last section sheds a light on the subject of polyomino covers that allow no jigsaw sudoku puzzles whatsoever.

The research presented in this document was done in pursuit of a Bachelor of Science degree from Leiden University, the Netherlands. It was supervised by Walter Kosters.

2 Dissecting the square

In how many ways could one dissect an $n \times n$ square into n polyominoes of size n? For $n = 1$ this is trivial (1), for $n = 2$ the answer would be 2 (dissections that are the same only when allowing rotations and reflections are considered distinct), $n = 3$ has 10 solutions (see Figure 3). With some patience the value for $n = 4$ was solved by hand on a whiteboard, resulting in 117 distinct configurations. With values quickly increasing, it became apparent that for finding more numbers in this sequence the use of a computer would be essential.

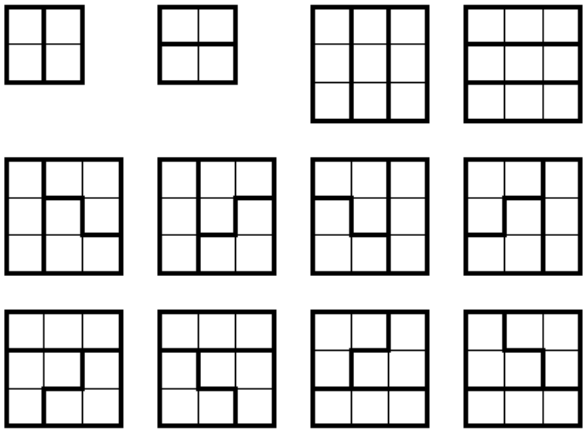

Figure 3: All polyomino covers for squares of size 2×2 and 3×3.

All throughout the approach described, the 64-bit integer datastructure unsigned long long, that is native to C++, was used to represent (sets of) polyominoes. The resulting restriction $n \leq 8$ was not a limitation within the scope of this project, because answers beyond this limit turned out to be out of reach regardless.

2.1 Generating the tiles

A *tile* will be defined as a polyomino with a fixed position in a given grid. This is convenient because the binary representations of tiles in a small grid, can be efficiently combined without repeated need for computationally intensive translations, rotations and reflections in the future.

In order to generate the required tiles efficiently, observe that any tile consisting of k unit squares, can in at least 1 way be described as the union of a tile of 1 unit square with a tile of $k-1$ unit squares (and specifically not necessarily in exactly k ways, since in some cases leaving out one unit square would leave the remainder disconnected). From this it follows that one can create the set of tiles of size n by starting with the empty set, iterating over the tile size $k \leq n$, starting at 1, and in each iteration building the set of tiles of size k from the set of tiles of size $k-1$ by for each tile t of size $k-1$, constructing the union with each unit square that is not in t, but does share an edge with t.

2.2 Areas modulo n

Not all tiles generated above are useful puzzle pieces: Some have small holes, some cause small areas to be created that can not be tiled (for a clarification, see Figure 4). To solve both these issues a depth first search can be done on the grid around each tile in the list generated, to discard those pieces that enforce an empty space on the grid with an area other than a multiple of n. For $n = 8$ this brought the number of tiles back from $64,678$ to $54,394$.

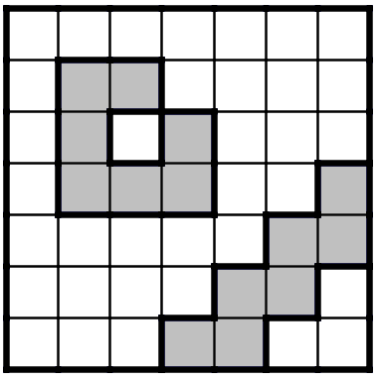

Figure 4: This 7×7 grid displays a tile with a 1×1 hole, which can not be tiled with polyominoes of size 7 and therefore will never render a valid dissection of the kind we want to count. The other tile shown, although it contains no holes, helps to create a secluded area of size 3 in the lower right corner that can not be tiled either. Both of these tiles can safely be discarded.

Recognizing areas that can not be tiled, simply because their size is not a multiple of n, also proved to be an important instrument for backtracking later on.

2.2.1 Useful holes

As an aside, polyominoes with holes in them are not useless by definition, but the smallest value of n for which a polyomino can disconnect its exterior from it's interior with the latter being of a size that is a multiple of n, is 23 (the multiple being 1). An example is shown in Figure 5.

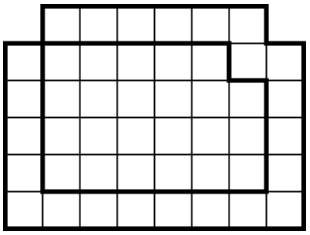

Figure 5: A polyomino of size 23 containing a hole of size 23; a smallest two dimensional cannibal.

One could also wonder how to solve this in another number of dimensions. A nonempty hyperpolyomino that disconnects its exterior from its interior, where the interior is exactly the size of the hyperpolyomino itself we will call a *cannibal*.

There is no one dimensional cannibal. In two dimensions the smallest cannibal is of size 23. The task to find the smallest three dimensional cannibal is not trivial and related work by Alonso and Cerf [4] on three dimensional polyominoes of minimal area seems to suggest analysis can be quite tedious.

The smallest three dimensional cannibal we found has size 139. The shape is based on an internal structure consisting of a $4 \times 5 \times 5$ block. This is extended on all 6 sides by one layer of size $(a - 2) \times (b - 2)$ for a side of size $a \times b$. Two opposite corners of the $4 \times 5 \times 5$ block are left out to allow one unit cube to connect 3 parts of the cannibal, rather than 2, on two occasions. Although this structure can contain 140 while being of size 139, it can be easily adapted to contain exactly 139. See Figure 6.

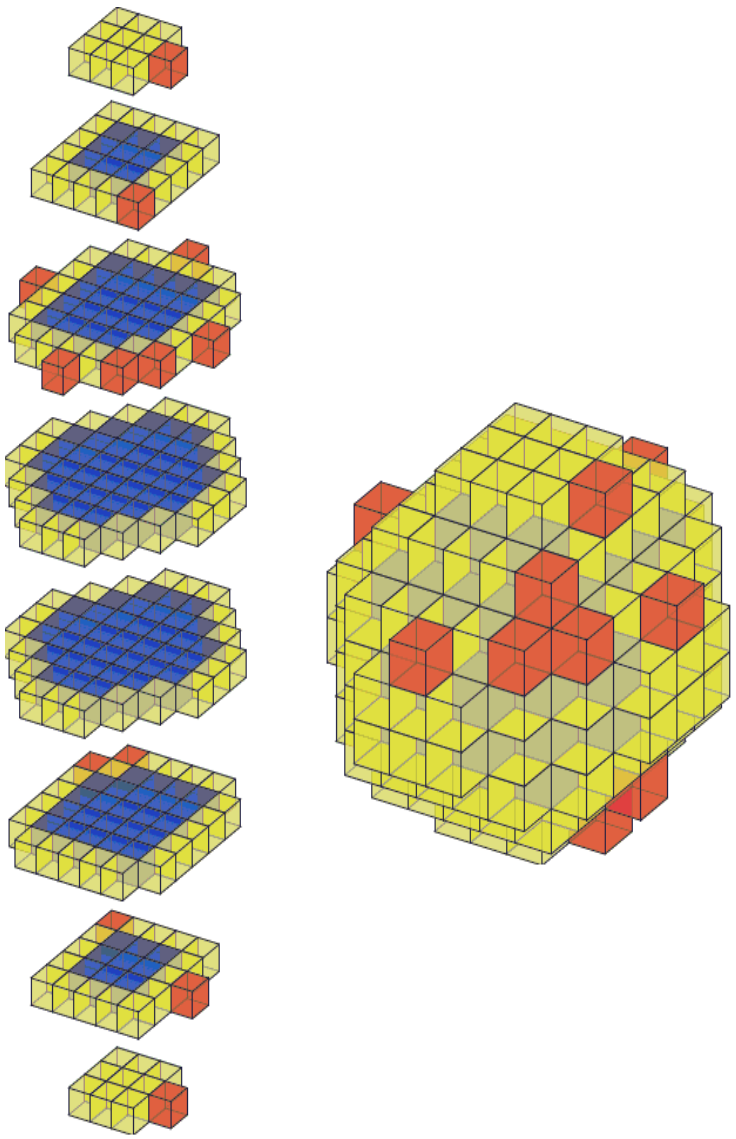

Figure 6: A three dimensional hyperpolyomino of size 139 containing a hole of size 140. With a minor adaptation it can be transformed into a smallest three dimensional cannibal of size 139.

2.2.2 Other useless areas

Not every area whose size a multiple of n is guaranteed to have a tiling with polyominoes of size n, the smallest counterexample being the T-tetromino, for $n = 2$ (Figure 7). However, for the values $n \leq 8$ such situations cannot be caused by only the border of the grid and a single polyomino and therefore this fact could not be exploited to further reduce the collection of tiles. The smallest n for which an example *was* found is 24 (see Figure 8). This search was not exhaustive.

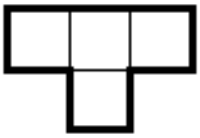

Figure 7: A T-tetromino can not be covered by two dominoes.

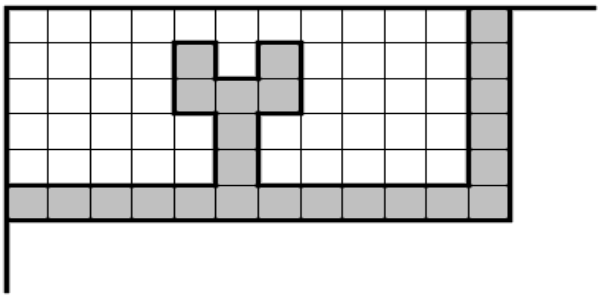

Figure 8: A polyomino of size 24, with the help of the border of a 24×24 square, enclosing an area of 48 that can not be tiled with polyominoes of size 24.

2.3 Deductions

Some tiles completely determine the presence of others, and this can be exploited. If for tiles t_1, t_2 the presence of t_1 implies the presence of t_2, then t_1 can be replaced by the combination of t_1 and t_2 (implemented as a bitwise *OR* of their binary representations), but tile t_2 should be preserved as is, since it might be used in configurations that t_1 is not part of. By enlarging tile $t1$, the algorithm will not as often be tricked into exploring partial

configurations that do contain t_1, but where some square that should be designated to t_2 is occupied by some other tile.

What is determined by a single tile is not strictly limited to one polyomino. For $n = 3$ and $n = 4$ there are even tiles that fix the entire grid by implying more than one other tile, as depicted in Figure 9. There is also a way to place a tile in a 4×4 grid such that the grid is not entirely fixed, but all possible resulting tilings are equivalent up to symmetries of the square (Figure 10). For $n \geq 5$ it is not possible for a single tile to determine the entire grid.

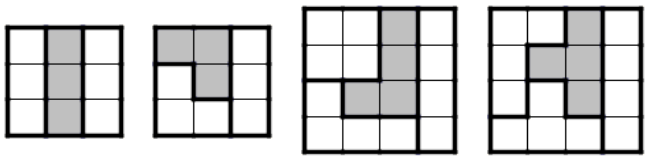

Figure 9: The polyomino tilings of the 3×3 and 4×4 square which are completely determined by a single tile, up to the symmetries of the square.

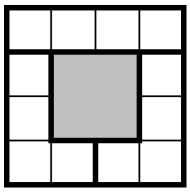

Figure 10: With a 2×2 square in the center, all tilings of a 4×4 grid are identical up to symmetries of the square.

2.4 Symmetry

The square has a symmetry group called the dihedral group D_4, containing 8 elements. That may sound like a quickly earned near factor 8 speedup (not exactly a factor 8, since some configurations are their own image under some transformations), but exploiting it wholly has some drawbacks.

To avoid generating configurations that are equal under transformation, one needs to have a good starting point. The only reasonable thing

to do seems to be to start with tiles touching the 4 corners of the square and to put restrictions on them. For example, give corner pieces an index based on their structure when rotated to the upper left corner (as a means of normalization). The piece in the upper left corner will be given the lowest indexed element of all, and among the two neighboring corners a $\leq$ relation will be obeyed. It becomes a bit more tricky because one has to consider $1 \times n$ pieces which touch two corners.

At the gain of a factor 8, this approach puts us at a depth of 4 in the intended depth first search of the problem space, but intuitively with fairly limited pruning possibilities. Also it would probably not work as well with the memoization optimization described in a later section.

Instead, we only exploited the symmetry about the main diagonal by choosing to first place tiles occupying the upper left corner and defining an order on them, combined with a partial exploitation of the horizontal symmetry by choosing the second tile from the set of tiles occupying the upper right corner (i.e., if the first piece didn't already reach there).

2.5 The order of evaluation

For starters, elements not occupying any of the two top corners were categorized on the least significant 1 in their bitwise representations and the new tile was chosen at the position of the least significant 0 in the (row major) representation of the current situation. Because this guarantees locality of placed tiles, this leads to heavy pruning.

Notice that this way of placing tiles is much smarter than the most straightforward way, where one would simply try every tile from the list, put it on there if it fits, and recurse with the index of the last tile as a parameter to fix the order and not count solutions $n!$ times due to all the different permutations.

Ignoring the pruning that is bound to happen, a quick analysis shows that the search tree for this straightforward method has $\frac{t^n}{n!}$ leafs, where t is the number of tiles and n the number of polyominoes needed to cover the grid. One should compare this to the method just described, where the worst case will have a uniform distribution of all tiles over n starting positions (which is extremely unrealistic, since there are about n^2 starting positions in total) which would give an upper bound of $(\frac{t}{n})^n = \frac{t^n}{n^n}$. Add to

this that pruning in the straightforward recursive way does not have the advantage of being vulnerable to exploitation of locality.

Intuitively when starting in the corner, because tiles there have limited space to move, they will probably collide more often, giving rise to more pruning. Experiments showed this was indeed the case, giving immediate speedups of about 12% for $n = 7$ and roughly 30% for $n = 8$ when defining an order based on this intuition by hand. To make this work, keep in mind the categorization should also depend on the order defined.

A *hill climbing* technique was used to establish an even better order of evaluation. Given the time needed at this point to perform the entire computation, $n = 6$ was the largest for which the hill climbing technique seemed reasonable, but it was hoped a pattern would emerge that could be extrapolated for larger values of n. For $n = 7$ tests were run with subsets of the collection of tiles for fine-tuning of the order.

With the square in the upper left corner fixed as the starting point, with most pieces put there occupying mostly squares above the main diagonal as a result on the order defined on them, it became apparent it was fruitful to fill the upper right quarter next (starting with the upper right corner, which is good since it was already decided upon) and then get back to finish the upper left quarter.

Additionally a small speedup was observed when not only classifying the tiles towards their earliest bit in the evaluation order defined, but also some information about their immediate neighborhood. More specifically we counted how many consecutive blocks extended from this earliest bit towards the right. Now when placing tiles in the recursion, whenever there are only, say, 3 empty consecutive empty positions, one does not have to attempt to place tiles there that require 4 or more.

2.6 The last tile

When there have been put $n - 1$ polyominoes on the grid, does one really need to have another loop to see if what remains is a tile that renders a solution? Simply checking if the remaining area is connected will not do, since it might represent a polyomino whose representing tile was enlarged with a tile it completely determined, during the deduction phase, and is no longer to be considered a valid tile in its original form.

Using a hash table containing every tile except the ones that were ruled out because they couldn't be part of any tiling (because they will never occur as the last missing piece) and those that were enlarged with a polyomino whose position they completely determined (because the region they fill might also be covered by another set of polyominoes that is not related by this kind of cause and effect, and, as we we have not counted in how many ways this can be done, we would be backtracking without counting all valid configurations), together with a suitable hash function, the check can be done in small constant time.

If the remaining part of the grid matches a polyomino in the hash, the index of the lowest indexed *monomino* (the proper term for a polyomino of size 1) in it (according to the order defined) is by definition higher than those of all other tiles put on the grid earlier, hereby obeying the constraint put in place to avoid counting different permutations rendering the same solution.

2.7 Memoization

Just as there are multiple ways to tile the $n \times n$ square, there are most often multiple ways to tile a part of the square. Storing the number of ways to finish the puzzle given a partly filled square when this number is known, allows for pruning once the situation reoccurs.

Quickly recognizing areas that can not be tiled, simply because their size is not a multiple of n, can be used to discard many configurations that would otherwise take up more time and space, only to evaluate to zero in the end anyway.

Trading space for time using a hash map for the memoization allowed for an enormous speedup, for $n = 8$ from just below 100 hours to about 10 minutes on a computer with 60 Gb RAM. Keeping track of the recursion depth allows for having a different map for every level, giving a small, yet significant, extra speedup (in the order of 10%).

2.8 An outline of the final algorithm

The outline of the algorithm is as follows:

```
int Count(partial square P)
{
  IF(P has been processed before)
    RETURN stored value;

  IF(P contains useless areas)
    RETURN 0;

  solutions = 0;
  B = the first location in P according to V;
  FOREACH(tile T indexed to B)
    IF(T is completely contained in P)
      solutions += Count(P - T);

  STORE(P, solutions);
  RETURN solutions;
}

int Dissections(int N)
{
  Generate set of tiles of size N;
  Enlarge tiles that enable deductions;
  Discard tiles that create useless areas;
  V = order in which to evaluate the locations;
  Index tiles to their first location according to V;
  Initialize the count of some partial squares
              in order to partially exploit symmetry;

  RETURN Count(NxN square);
}
```

2.9 The sequence

The resulting number of dissections $t(n)$ of the $n \times n$ square into polyominoes of size n can be seen in Table 1. The third column contains a proposed normalization for the size of the grid which might help to predict larger values of $t(n)$ in the future. The near straight line in Figure 11 seems to support the applicability of this normalization.

n	$t(n)$	$t(n)^{1/n^2}$
1	1	$1.0000\ldots$
2	2	$1.1892\ldots$
3	10	$1.2915\ldots$
4	117	$1.3467\ldots$
5	4,006	$1.3935\ldots$
6	451,206	$1.4357\ldots$
7	158,753,814	$1.4702\ldots$
8	187,497,290,034	$1.5002\ldots$

Table 1: The number of tilings of the square $t(n)$.

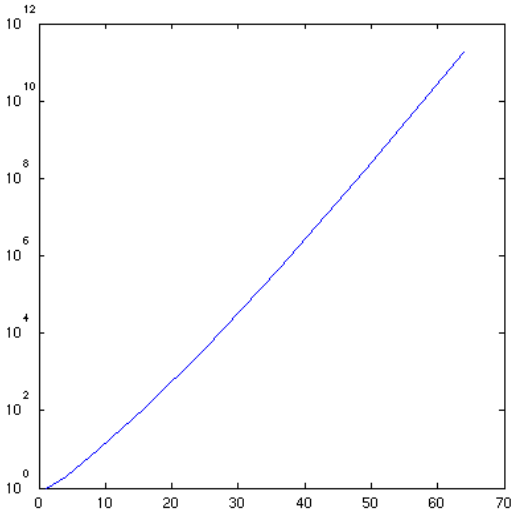

Figure 11: A graph depicting $t(n)$ on a logarithmic scale on the vertical axis versus n^2 on the horizontal axis; a near straight line.

The On-Line Encyclopedia of Integer Sequences does now refer to t as $A172477$ [5]. It intersects with integer sequences found before by Ron Hardin, who investigated the more general problem of tilings of rectangles with polyominoes of equal size. Our results extend his findings; in particular $t(8)$ is an extension to $A167258$, the number of tilings of an $n \times 8$ rectangle with polyominoes of size 8.

2.10 Comparison with Dancing Links

Dancing links [6] is the name of a technique suggested by Donald Knuth to efficiently implement his *Algorithm X*. This Algorithm X is a backtracking algorithm that finds all solutions to the *exact cover problem* [7]. Tiling the square with polyominoes is an instance of the exact cover problem, so it makes sense to compare results. We used a clean existing implementation [8] of dancing links and we fed it the tiles that do not create any regions that do not have an area that is a multiple of n.

Our approach outperforms dancing links by far in this particular domain, at least within the scope that both algorithms can deal with the current state of technology. The amount of states that we can memoize is limited by the size of the computer memory. Beyond this limit it is a whole different race, but before we reach this limit we already hit the ceiling on what we can compute in limited time with limited resources.

If we disregard the time required to generate the tiles, for $n = 6$ our method is approximately 350 times faster. For $n = 7$ the factor is even larger. So how come we beat dancing links within its own domain? It is mainly because the memoization (and, to a lesser extent, exploiting symmetry) allows us to count the number of dissections without actually realizing every single one of them. If you know you have 15 pairs of socks, you don't need to count every single sock to know there are 30 socks in total.

3 Counting jigsaw sudoku puzzles

To count all completed jigsaw sudoku grids one could either generate the possible tilings of the square and fill them with digits obeying row, column and shape constraints, or one could generate all Latin squares and count the possible valid tilings. The latter seems to best fit the method to generate tilings, as developed in the previous chapter, since a given Latin square will render many more tiles invalid than those who were already discarded, namely those which cover some digit more than once. Additionally, generating the Latin squares beforehand, this time allows to *fully* exploit the symmetries of the square.

3.1 Reducing Latin squares

Relabeling the numbers in a Latin square has no effect on the number of distinct jigsaw sudokus that can be based on it. Therefore, any Latin square can be relabeled such that the upper row contains its elements in increasing order and Latin squares which are equal after applying this *normalization* operation, contribute the same number of solutions to the total count. Since there are $n!$ permutations of n elements, every Latin square with the top row in sorted order in actuality represents $n!$ Latin squares with the same number of jigsaw sudokus based on it, meaning it suffices to only generate all Latin squares with the top row in sorted order and multiply the answer by $n!$, accounting for a speedup of $n!$.

Contrary to what can be done when counting Latin squares or (to some extent) in counting the number of standard sudokus, it is not true in general that the rows can be sorted in increasing order of their leftmost element (which would have contributed another factor $(n-1)!$ speedup) without altering the result. This is because contrary to relabeling numbers, reordering rows can fundamentally change the structure of jigsaw sudoku puzzles (for an example see Figure 12). The proper terminology for a normalization of a Latin square such that both the top row (either by column reordering or by relabeling) and the leftmost column (by row reordering) are in sorted order is *reduction*. To denote the normalization by relabeling only, we will use the term *semi reduction*.

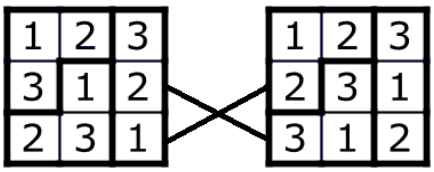

Figure 12: In this normalized jigsaw sudoku (on the left) sorting the rows by their leftmost element results in an invalid jigsaw sudoku grid.

3.2 Symmetry

Up to 8 semi reduced Latin squares can be equivalent under a symmetry of the square if we once again allow for normalization by relabeling. We first generated all $b(n)$ borders of $n \times n$ squares satisfying Latin square constraints (Figure 13) with the top row in sorted order, each edge containing a permutation of the numbers 1 through n and no conflicts between elements that were on the same row or column but on opposite edges of the square.

1	2	3	4	5
4				2
5				3
3				1
2	5	1	3	4

Figure 13: An example of a 5×5 square border satisfying Latin square constraints.

After an order has been defined (for example the lexicographical order of the border elements in clockwise direction starting in the upper left corner), the representative of such a border can be found by creating all 8 symmetries, normalizing them and picking the smallest one according to the defined order. A hashmap can be used to keep count of how many borders each of the $br(n)$ representatives actually represents.

Table 2 quantitatively summarizes what the effect of the mentioned optimizations is in practice. As could be expected, the ratio $br(n)/b(n)$ turns out to approach 8 from below. The number $els(n)$ of Latin squares that

n	$b(n)$	$br(n)$	$els(n)$	$ls(n)$
1	1	1	1	1
2	1	1	1	2
3	2	1	1	12
4	26	11	12	576
5	924	139	205	161,280
6	81,624	10,704	150,580	812,851,200
7	13,433,520	1,683,091	1,528,450,336	61,479,419,904,000

Table 2: The number of valid normalized borders $b(n)$ for a given size of squares n satisfying Latin square constraints, the number of distinct border representatives $br(n)$ after exploiting the symmetry of the square, the number of Latin squares that still have to be evaluated $els(n)$, and for comparison also the number of Latin squares $ls(n)$.

still have to be evaluated is much smaller than the total number of Latin squares $ls(n)$ of size n.

Note that the partial exploitation of the symmetries of the square as had been done in tiling the square in the previous chapter, is now no longer allowed.

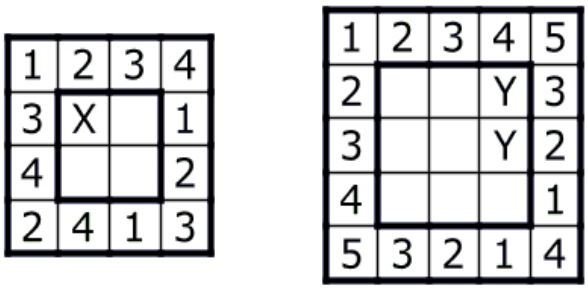

Figure 14: The border of the 4×4 square obeying Latin square constraints shown on the left can not be completed into a Latin square, because the position marked with X is not allowed to be any of the numbers 1, 2, 3 and 4. The border of the 5×5 square obeying Latin square constraints shown on the right can not be completed into a Latin square, because the positions marked with Y would both be forced to be a 5.

It would be a mistake to think that every border satisfying Latin square constraints can be completed into a Latin square. Figure 14 shows exceptions for borders of the 4×4 and 5×5 square.

The value $b(n)$ also applies for other subsets of two rows and two columns of the square.

3.2.1 A formula for $b(n)$

A *derangement* is a permutation that leaves none of the elements invariant [9]. Their number d_n for length n, can be computed recursively as follows: $d_0 = 1$ and for $n > 0$ it holds that $d_n = nd_{n-1} + (-1)^n$. Another recurrence relation, one that can help simplifying later on, is: $d_0 = 1$, $d_1 = 0$ and for $n \geq 2$ it holds that $d_n = (n-1)(d_{n-1} + d_{n-2})$. For $n \geq 1$ a closed formula also exists: $d_n = \left\lfloor \frac{n!}{e} \right\rceil$, where the bracket notation means rounding to the nearest integer.

We distinguish 4 cases, based on whether the content of the lower left corner is or is not equal to n and whether the content of the lower right corner is or is not equal to 1, as indicated in Figure 15. We explicitly assume $n \geq 4$.

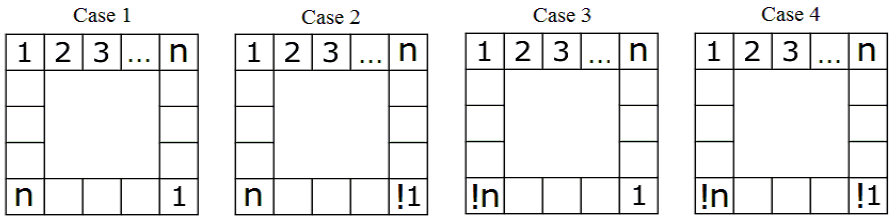

Figure 15: To derive a formula for $b(n)$ we distinguish 4 cases. An exclamation mark is used to denote that the particular digit behind it is not allowed at that location.

Case 1 allows for the remaining $n-2$ elements of the leftmost column to be filled the digits $\{2 \ldots n-1\}$ without any restrictions. The remainder of the rightmost column should also be filled with this set of digits, but because of the Latin square requirements that state two digits on the same row or in the same column should be different, it can only be completed with derangements of what was entered on the left. The empty cells on the bottom row should become a derangement of the permutation of $\{2 \ldots n-$

$1\}$ as it occurs above them on the top row. As a result Case 1 amounts to $(n-2)!d_{n-2}^2$ ways to complete borders with the first row in sorted order.

Case 2 can have any of $n-2$ different values in the lower right corner. Again the $n-2$ elements remaining to be filled in in the leftmost column can be entered in any order. Now both the remainder of the rightmost column and the remainder of the bottom row share $n-3$ digits with their counterparts respectively on the left and above, and they both have one digit which does not occur in their counterpart (*misfits*), likewise their counterparts have one digit they don't have. If for each we only consider all d_{n-2} derangements while temporarily regarding misfits to be of the same digit class, we forget to take into account the d_{n-3} completions where the misfits match while nothing else does. As a result Case 2 amounts to $(n-2)!(n-2)(d_{n-2}+d_{n-3})^2$.

Case 3 is symmetrical to Case 2 and as such results in the same amount of configurations to be counted.

In Case 4 the bottom corners can be any distinct values from the set $\{2\dots n-1\}$, so there are $(n-2)(n-3)$ ways to select them. Once again the remainder of the leftmost column can be completed in any of $(n-2)!$ ways. As in Case 2 there is a discrepancy between the sets of digits we called counterparts; this time there are 2 misfits to deal with per set. Let's consider the ordered pairs of misfits (a_1, b_1) for one set and (a_2, b_2) for its counterpart. The number of reorderings such that $a_1 \neq a_2$ and $b_1 \neq b_2$ is simply d_{n-2}. If only one of these inequalities holds, the number of reorderings is $d_{n-3}+d_{n-4}$. However, if we add these values we counted the number of reorderings such that neither inequality holds twice and so we need to compensate for that by subtracting d_{n-4}.

This results in the formula:

$$b(n) = (n-2)!\left(d_{n-2}^2+2(n-2)(d_{n-2}+d_{n-3})^2+(n-2)(n-3)(d_{n-2}+2d_{n-3}+d_{n-4})^2\right)$$

Or, equivalently:

$$b(n) = (n-2)!\left(\frac{n-1}{n-2}d_{n-1}^2 + 2d_{n-1}d_{n-2} + \frac{2n-5}{n-3}d_{n-2}^2\right)$$

If one insists on having a formula without derangements, it is possible to substitute d_k by $\left\lfloor \frac{k!}{e} \right\rceil$ (but note that factorials and derangements could as well be considered of equal elegance, since the recurrence relations by which they are defined are very similar):

$$b(n) = (n-2)!\left(\frac{n-1}{n-2}\left\lfloor\frac{(n-1)!}{e}\right\rceil^2+2\left\lfloor\frac{(n-1)!}{e}\right\rceil\left\lfloor\frac{(n-2)!}{e}\right\rceil+\frac{2n-5}{n-3}\left\lfloor\frac{(n-2)!}{e}\right\rceil^2\right)$$

The On-Line Encyclopedia of Integer Sequences now refers to b as $A173103$ [5] and to the total number of possible borders obeying Latin squares constraints (multiplying by $n!$) as $A173104$.

3.3 Latin square representation

Rather than as arrays, Latin squares can be represented by a set of n bitmasks, each denoting the placement of digits of the same kind. Starting with filled-in borders the remainder can be generated position by position in row major order. This way a precalculated set of n^2 bitmasks corresponding to a row and a column of binary 1's allows for checking in constant time if a specific digit will fit at his target position.

3.4 Selecting the tiles

For every Latin square that is going to be evaluated a subselection of the set of all tiles as generated in Section 2.1 should be made, selecting only those tiles that cover a set of n distinct digits. Using the preprocessed bitmasks indicating the positions of identical digits within the Latin square, the selection can be done very efficiently. Since this is not a bottleneck in the program however, there is no real need for heavy optimizations here.

The deduction process from Section 2.3 was dropped to facilitate a straightforward bitwise implementation, but for larger values of n the deduction method might still be worth using, either right after generating the general set of tiles or after making a selection of tiles. The different options represent different trade-offs regarding the amount of time spent cutting tile sets and the speedups resulting from them and it is not obvious without further investigation what the best choice would be.

3.5 The order of evaluation *(continued)*

When we used a hill climbing technique to establish a better order of evaluation of the positions in the grid regarding the indexing of the tiles, in Section 2.5, we noticed a severe impact of the partial exploitation of the symmetries of the square on the resulting order. In the current situation we do not perform this partial exploitation, and it is beneficial to apply hill climbing once more within this new environment.

In counting the number of jigsaw sudoku solutions, the set of valid tiles differs from one subproblem to another, since different Latin squares impose different constraints, but because these extra constraints render many more tiles invalid, leading to much faster program termination, and because of the very large number of semi reduced Latin squares, leading to much slower program termination, the emphasis has partly shifted to dealing with a great number of subproblems. As a consequence, it does not seem feasible to perform hill climbing to find a suitable evaluation order for every single set of tiles we encounter. Instead, we performed the hill climbing technique for a random sample of Latin squares and from the results obtained for $n = 6$, once again for $n = 7$ an order was extrapolated and then refined with more hill climbing.

3.6 The sequence

Table 3 contains the number of completed jigsaw sudoku puzzles, $js(n)$, in the rightmost column. It also shows the number of tilings of the square and the number of Latin squares of order n, $t(n)$ and $ls(n)$ respectively. We were able to compute $js(6)$ in under 500 seconds, whereas $js(7)$ took already 273 hours of computation time. Luckily the algorithm lends itself to be distributed and could be run on 4 processors simultaneously.

n	$t(n)$	$ls(n)$	$js(n)$
1	1	1	1
2	2	2	4
3	10	12	72
4	117	576	13,872
5	4,006	161,280	11,762,160
6	451,206	812,851,200	234,312,972,480
7	158,753,814	61,479,419,904,000	41,182,101,508,222,080
8	187,497,290,034	108,776,032,459,082,956,800	?

Table 3: The three quantities $t(n)$, $ls(n)$ and $js(n)$.

The values for $js(n)$ for $n \leq 6$ were independently verified with the reverse approach: For every tiling of the square, every semi reduced Latin square obeying the constraints induced by the shape of the tiles was generated and counted.

The On-Line Encyclopedia of Integer Sequences now refers to js as $A172478$ [5].

4 Minimal jigsaw sudoku puzzles

To find the minimal number of givens needed to enforce a unique sudoku puzzle solution for different polyomino tilings (the size of their *smallest critical sets*), for each tiling all Latin squares obeying the constraints determined by the polyomino tiling at hand can be generated, and subsequently iterations can be made over subsets of locations of this jigsaw sudoku puzzle, and then all the selected Latin squares can be classified according to their content at these locations. If any of these classes contains exactly one element, the combination of this polyomino cover, this subset of locations and this Latin square is a valid jigsaw sudoku puzzle with a unique solution. If all subsets of smaller size have already been discarded, then the resulting puzzle has a minimum possible number of givens for this particular polyomino tiling.

Determining a smallest critical set for a single polyomino tiling this way, very roughly stated, has a time complexity of

$$O\left(\sum_{m=n-1}^{|SCS|}\binom{n^2}{m}\times ls(n)\times(m+\log(ls(n)))\right)$$

in which $|SCS|$ is the size of the smallest critical sets and the factor $m+\log(ls(n))$ stems from inserting the appropriate data for a Latin square into some tree-like datastructure.

4.1 Sorting the top row

It might be somewhat surprising that the set of potential solutions with the top row of numbers in sorted order is sufficiently large for the purpose of finding the smallest critical sets for a given shape; there is really no need to (repeatedly) evaluate the earlier proposed complete set of Latin squares, which is a factor $n!$ larger.

Why is this? First note that only solutions for which the by the subset of locations induced sequence of elements (the *pattern*) contains at least $n-1$ different symbols (the *rich patterns*), are to be considered, because others do not enforce unique solutions (for any solution to a jigsaw sudoku puzzle with less than $n-1$ different symbols given, the symbols that did not occur among these givens can be permuted, leading to other solutions).

With this requirement imposed on the set of givens, we know that any valid solution to a certain polyomino cover is not a solution anymore after applying any permutation to the alphabet of numbers except the identity, since there is no permutation besides the identity that leaves $n - 1$ distinct elements in place (and if there is one thing we know about givens, it's that they stay as they are). In other words, with such a set of givens, for any Latin square there is either one solution among the Latin squares equivalent under permutation of the alphabet, or none at all.

Let a *normalized pattern* be a pattern whose representation is the lexicographically smallest of all of its relabelings. Trivially, for a given subset of locations, all Latin squares within one equivalence class have the same normalized pattern. If a pattern in one equivalence class equals a pattern in another, also their normalized patterns will be equal. It follows that the number of Latin squares that match a certain rich pattern, equals the number of equivalence classes for which the normalization of the pattern induced by the same subset of locations equals said pattern when normalized. Therefore, to determine if the subset of locations at hand gives rise to a puzzle with a unique solution, it suffices to aggregate the normalized rich induced patterns of all semi reduced Latin squares and check if any of them occurs exactly once.

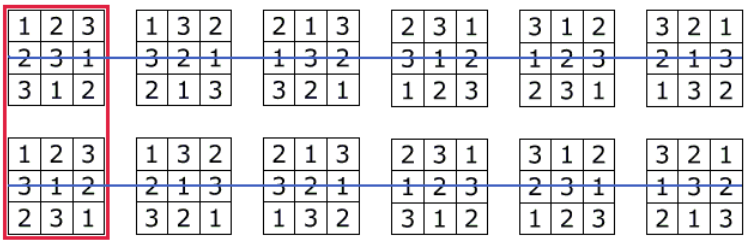

Figure 16: A structured view on the set of 3×3 Latin squares. The set of semi reduced Latin squares is contained in a red rectangle which spans the leftmost column. Latin squares are on the same row if and only if they are equivalent up to relabeling.

Figure 16 shows a structured view on the set of all Latin squares for $n = 3$. For a given subset of locations, any two Latin squares on the same row give rise to the same normalized pattern, yet if the induced patterns are rich, they are distinct. The leftmost column shows the semi reduced Latin squares: the representatives of the equivalence classes under relabeling.

4.2 Flirting with binary search

In the complexity estimation above, it was assumed that the cardinality of the subsets is incremented by 1 after all subsets of the current size have been dealt with.

Wouldn't it be smarter to do a binary search over the range $1 \dots n^2$, or even better, over the range $n-1 \dots (n-1)^2$? This suggested lower bound $n-1$ stems from the fact that since less than $n-1$ givens means there are less than $n-1$ different symbols among them, and as we have decided before, this does not yield puzzles with a unique solution. The upper bound $(n-1)^2$ can be justified by noticing that regardless of the polyomino tiling, a valid solution can be retrieved after the bottom row and the rightmost column have been erased.

However, in reality the size of the smallest critical sets most often seems to be very close to the lower bound $n-1$ (at least for the values of n that were within our reach), and because of the very rapid growth of the amount of subsets of equal size when this size increases, it is much more efficient to increment the cardinality of the subsets by 1 starting from $n-1$ and to stop as soon as something useful is found.

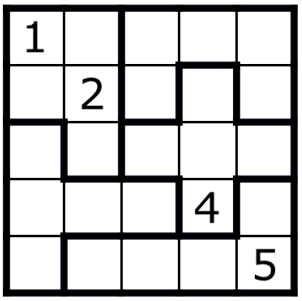

Figure 17: A homemade 5×5 jigsaw sudoku with only 4 givens [12].

It can be shown that for any n there exists a polyomino cover which requires only $n-1$ givens to ensure a unique solution [10]. Figure 17 shows an example of such a puzzle with only $n-1$ givens. Furthermore it has been conjectured the size of the smallest critical sets for a Latin square is $\lfloor \frac{n^2}{4} \rfloor$ [11].

4.3 Results

Table 4 shows the number of polyomino tilings $t(n)$ categorized according to the size of their smallest critical sets. We observe that already for $n = 4$ there are polyomino tilings that need n givens and for $n = 5$ there are tilings that require more than n givens.

n	∞	0	1	2	3	4	5	6	7	8	9
1	-	1	-	-	-	-	-	-	-	-	-
2	-	-	2	-	-	-	-	-	-	-	-
3	-	-	-	10	-	-	-	-	-	-	-
4	8	-	-	-	100	9	-	-	-	-	-
5	64	-	-	-	-	3,740	200	2	-	-	-
6	17,156	-	-	-	-	-	392,380	39,824	1,772	72	2

Table 4: The number of polyomino covers of size n split based on the size of their smallest critical sets; ∞ denotes the number of polyomino covers that allow no jigsaw sudoku solutions at all. Blanks in the table are denoted by dashes.

We also observe that the 2 tilings for $n = 5$ that require 6 givens and the 2 tilings for $n = 6$ that require 9 givens, are the tilings consisting of only rows or only columns. These tilings do actually not impose any extra constraints at all, and in general their smallest critical sets are equal to the smallest critical sets for Latin squares of the same size. Our results are in accordance with those of Bate and Van Rees [13], who computed the size of the smallest critical set of Latin squares of size 5 and 6 by other means.

There is a tendency for less irregular shaped tilings to have larger smallest critical sets, largely due to symmetries that need to be broken to enforce uniqueness. Those who require 8 givens for 6×6 puzzles, for example, all mostly consist of lines and/or 2×3 rectangles.

For $n = 4$ those polyomino covers who require 4 givens are exactly those who consist of only lines and 2×2 squares.

4.4 Early retirement

We know that the size of the smallest critical sets of the Latin square of order n is an upper bound on the smallest critical set of each and every

jigsaw sudoku puzzle of order n. This can be used by first determining the size of the smallest critical sets of the Latin square, perhaps by other means, and concluding that a tiling belongs in this category if no smallest critical set has been found among the smaller sizes.

4.5 Derived logic

By simple logic one can often find identities between the content of different squares in a puzzle, given the polyomino cover (for example consider the situation where a polyomino covers all but one squares in the first column, then the excluded square equals the square in the polyomino that is not in that column). More complicated logic will often reveal more identities. However, with all solutions to a polyomino cover listed, we can just check if any potential identity holds in all of the valid solutions and thereby reveal every single one of them without further ado.

When we find sets of squares with identical content, it is a waste of resources to have the subset iteration cover more than one of them.

4.6 Symmetries of the square

Since the number of polyomino covers for $n = 6$ is only $451,206$, it is feasible to consider every rotation and reflection of each of them and use a suitable normalization and order to identify a unique representation for each class of polyomino covers equivalent under symmetry of the square. If the positions in the polyomino cover representation are now numbered with the index of the polyomino they belong to for the sake of being able to distinguish them, once again as a way of normalization we can require every newly used element to be translated into the lowest numbered unused symbol, after which the order can be defined by considering the value represented by concatenation of the values at every position in row major order.

By only computing the minimal number of givens needed for the representation of each equivalence class, we can expect to gain a near factor 8 reduction in the number of cases to process for larger values of n and accordingly a near factor 8 speedup of the algorithm.

4.7 Cutting edge symmetries

There is another trick up our sleeves. Consider a polyomino cover in which one can cut the grid along one or more parallel lines that do not intersect any polyomino. Putting the pieces back together in a different order, or with one or more of them reversed around an axis parallel to the cuts (as illustrated in Figure 18), does not change the sudoku requirements on the rows parallel to the cuts, nor does it change the sudoku requirements on the rows perpendicular to the cuts, therefore the size of the smallest critical sets will remain equal and we can expand our notion of equivalence class accordingly. We can further reduce the number of cases we need to put to the expensive test of finding the size of the smallest critical sets. Since a higher degree of symmetry will often lead to a higher number of required digits, this optimization can be expected to have a larger impact exactly on the more expensive cases where optimization is most welcome. For example, consider the polyomino covers that resemble a cover consisting of polyominoes each spanning a single row, but differ slightly. These are expected to have a relatively large smallest critical set as they do not impose many requirements other than those of the Latin square, and, as it happens to to be, larger smallest critical sets take longer to find. At the same time such polyomino covers typically have more lines along which one can cut them in order to reorder the pieces and more such reorderings.

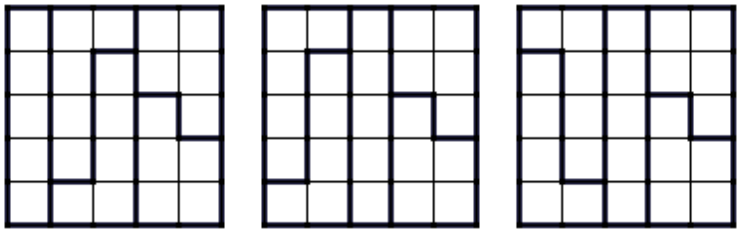

Figure 18: An example of cutting edge symmetries: Three polyomino covers that are equivalent under cutting along parallel lines and glueing the resulting pieces back together in some order with some possibly mirrored in a line parallel to the cuts, regarding the size of their smallest critical sets.

To find the set of polyomino covers with a single representative for each equivalence class of the cutting edge symmetries, once again we use a normalization with respect to the symmetries of the square and this

time instead of normalizing the numbers indicating the value in the Latin square, we normalize the numbers identifying the polyomino they represent within the polyomino cover.

For every polyomino cover to consider we do a *breadth first search* (*BFS*) to find the normalized representation denoting the representative of its equivalence class. Starting with its original representation on a queue, it branches by first establishing where the parallel cuts can be made. This can be done simply by evaluating column by column and for each polyomino keeping track of how many of its squares have been seen and also keeping track of of how many polyominoes a multiple of n squares have been seen (the number of squares seen for each polyomino and the number of them who are a multiple of n, say m, can be updated in constant time for each of the n^2 squares). Whenever m equals n after evaluating an entire column, we have found a place where a cut can be made (but we will ignore the trivial cuts on the sides). The pieces we then permute and reflect, and if we have not seen them in the current BFS, we add them at te end of the queue. We need to do the same for the transpose within the BFS, because some tilings have cuttable edges in two directions. The number of tilings each representative represents is counted in a tree-like structure (or a hashmap).

Where similar shapes are glued together there is room for some optimization, since their reordering leaves the result unchanged.

4.8 Exotic symmetries

Notice that in fact any reordering of the rows or columns that leaves every polyomino connected, will do (*exotic symmetries*), so we can once again relax the definition of our equivalence class. For an example, see Figure 19.

In fact the reduction holds for any permutation of the small squares that is structurally isomorphic, i.e., the constraints induced by the polyominoes, rows and columns are identical for some permutation of the small squares, but we did not investigate if this leads to a reduction of the number of polyomino covers to actually evaluate. In the most straightforward way it would be very computationally intensive to establish the isomorphism, so a more clever algorithm would be required to make the effort worthwhile.

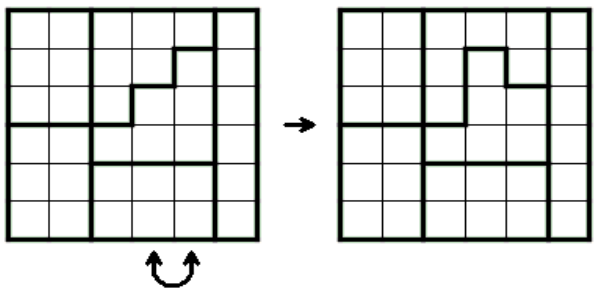

Figure 19: Two polyomino covers that are equivalent under reordering rows and reordering columns while leaving all polyominoes connected regarding the size of the smallest critical sets.

To look at all of the exotic symmetries, we generated all $(n!)^2$ permutations of the rows and columns of the square. We can save a factor 4 here by picking two rows and two columns that will have to occur in a given order, so as not to generate exact horizontal and vertical mirror images that are reflected in the normalization step anyway.

When the rows have been permuted, there is room for backtracking in permuting the columns since we can often know beforehand something has already become disconnected. In this case we can not merely do the same for the rows, since an action disconnecting a polyomino in permuting the rows, might be followed by some permutation on the columns reassembling it.

A useful insight is acquired by considering the projection of the polyominoes on the x-axis and on the y-axis. The permutations of rows and columns in this regard *are* independent and it is required (but not sufficient) that every projection remains connected. This can serve as a source of backtracking in a depth first search approach.

Table 5 tabulates the results.

4.9 Subset evaluation optimizations

By far the most computationally intensive is the evaluation of those tilings that have many row spanning polyominoes (in the normalized representation there are no row spanning columns), because they tend to have larger

n	$t(n)$	$sq(n)$	$c(n)$	$x(n)$
1	1	1	1	1
2	2	1	1	1
3	10	2	2	2
4	117	22	15	15
5	4,006	515	339	243
6	451,206	56,734	43,871	28,659

Table 5: The number of polyomino covers of the $n \times n$ square $t(n)$ compared with the number of equivalence classes regarding the symmetries of the square $sq(n)$, the number of equivalence classes under cutting edge symmetries $c(n)$ and finally the number of equivalence classes under exotic symmetries $x(n)$.

smallest critical sets. Interestingly there is an optimization which targets exactly these grids: We can define an order on the subsets of the contents of the row spanning polyominoes that belong to the subsets checked for being critical sets. The easiest way would be to convert these subsets to n-digit binary numbers and to say that for subset a to be above subset b, $a \leq b$ should hold. For r of these rows, this effectively reduces the amount of work by a factor $r!$. Furthermore, at most one of them is allowed to be associated with the empty set, since any solution is supposed to be unique while having two empty row polyominoes would allow for swapping their (non identical) contents.

To a lesser extent the same tricks can also be used on repeated structures spanning multiple complete lines. As a side effect of the normalization process these repetitions are grouped together in the same orientation, although they do need to be discovered first. This can be done by checking a row spanning structure that can not be decomposed into more than one nonempty row spanning structures against the previous one.

Finally, it appears a similar order can be imposed on the contents of identical columns, however, on the columns by identical we mean something quite different: Each of their squares should be part of the same polyomino as the one at the same height in their identical counterpart. As a matter of fact, this also applies to some of the rows! However, we can not just blend this in, since these orthogonal orders are not independent, but still there should exist a more elaborate method to deal with it.

Some polyomino covers are their own image under symmetries of the square. Also this can be worth exploiting, because we should keep in mind for each subset the operation of determining whether it can be the basis of a smallest critical set is quite expensive.

As a more general remark, keeping in mind how many elements are currently in subsets and how many will at least be needed in the rest of the grid, can help to backtrack while generating the candidate smallest critical sets much more often.

Some squares contain an identical value due to the requirements of the polyomino cover. Of course for each of these sets at most one element will be in a smallest critical set, but we can not merely decide which one this should be beforehand, because it might interfere with other optimizations.

A very small optimization is to never attempt to have n givens on a row, in a column or in a polyomino.

4.10 Substructures

When considering the structure imposed on a jigsaw sudoku puzzle by its polyomino cover, it becomes apparent that some of these requirement structures are embedded in others.

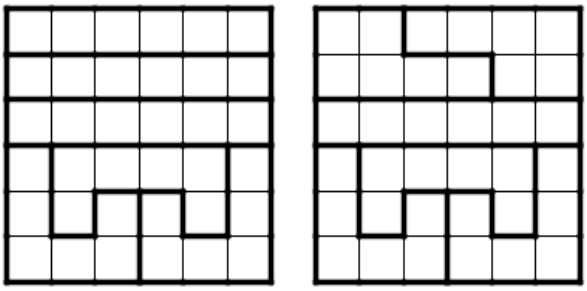

Figure 20: The requirements structure imposed by the leftmost polyomino cover is embedded in the one imposed by the polyomino cover on the right.

In Figure 20, for example, the rightmost polyomino cover imposes all

the requirements on the Latin squares that the leftmost polyomino cover imposes, and some more. Therefore we know that the size of the smallest critical sets for the leftmost tiling is at least as large as the size of the smallest critical sets for the rightmost tiling, so by evaluating them in a topologically sorted order we can cut costs.

5 Impossible jigsaw sudoku puzzles

5.1 Small examples

As apparent from Table 4, not every polyomino cover can be used for a jigsaw sudoku puzzle. In contrast, any Latin square *can* be the solution to some jigsaw sudoku puzzle, since there is always the trivial polyomino cover in which each polyomino covers exactly one of the rows of the Latin square, and from any Latin square one can erase the first row and the first column without harming increasing the number of solutions.

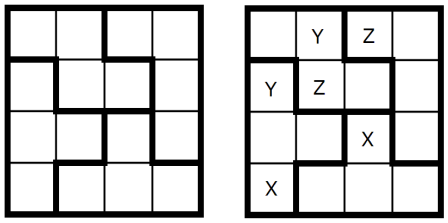

Figure 21: The smallest polyomino cover that can not be used to make a jigsaw sudoku puzzle.

The smallest n for which there exists a polyomino cover that can not be used to make a jigsaw sudoku puzzle is 4. In Table 4 it is written that there are 8, but these are all equivalent under symmetry of the square. Figure 21 shows one orientation.

To quickly determine that this polyomino cover can indeed not be used to make a jigsaw sudoku puzzle, note the following: The two letters X denote identical digits, because the bottom row and the T-tetromino in the bottom right both contain the numbers 1 through 4 and they overlap for three squares. Therefore the remaining ones (denoted by X) must be identical. The two letters Y denote identical digits because the Y outside the S-tetromino is not allowed to be identical to all numbers within the S-tetromino but the one with Y in it. Exactly the same holds for the two numbers denoted by Z. Furthermore, X, Y and Z are pairwise distinct, since each pair occurs somewhere in a row or column. The problem is that apparently the number denoted by X can not occur in the second column and this contradicts the definition of a Latin square.

For $n = 5$ Table 4 says there are 64 polyomino covers that no jigsaw sudoku puzzles can be based on. Once again we can disregard many of them because of the symmetries of the square and we are left with the 8 configurations shows in Figure 22.

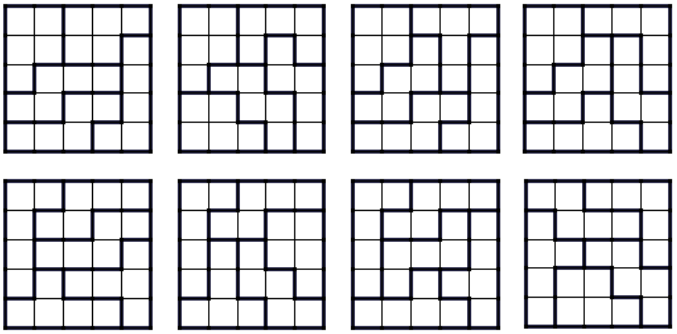

Figure 22: Polyomino covers of size 5×5 that can not be made into jigsaw sudoku puzzles.

5.2 A general proof

We can prove there are infinitely many polyomino covers that do not allow any jigsaw sudoku puzzle solution. Even stronger: We can prove for any $n \geq 4$ there exists a polyomino cover that does not allow any jigsaw sudoku puzzle solution. Figure 23 serves to illustrate the proof to follow.

Assume $n \geq 4$. The location of an element on the ith row in the jth column of a jigsaw sudoku puzzle is denoted by $[i, j]$, and (i, j) refers to the element at position $[i, j]$. We will use the notation P_k ($1 \leq k \leq n$) to identify a polyomino in the polyomino cover and it is a union of element positions.

Let $P_1 = \bigcup_{y=1}^{n-1}[y, 1] \cup [1, 2]$.
For $2 \leq k \leq n-2$ let $P_k = \bigcup_{x=k+1}^{n}[k-1, x] \cup \bigcup_{x=2}^{k+1}[k, x]$.

It can be easily verified that each of these is truly a polyomino, i.e., it consists for n distinct squares and it is connected. Also one can without much effort demonstrate that these polyominoes do not overlap, i.e., the intersection $P_a \cap P_b \neq \emptyset$ if and only if $a = b$.

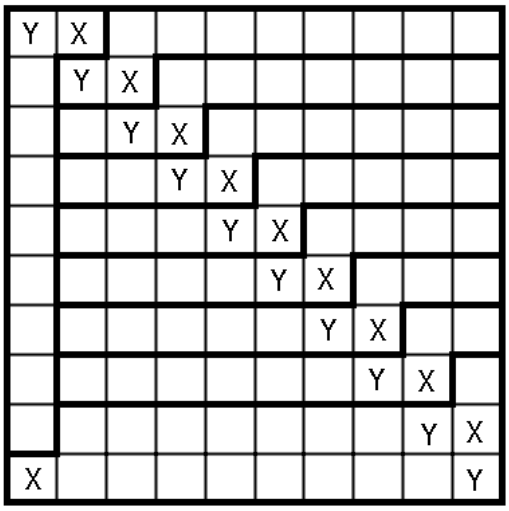

Figure 23: An illustration of the proof that for any $n \geq 4$ there exists a polyomino cover for which there exists no jigsaw sudoku puzzle solution. In this example $n = 10$.

Now call $(n, 1) = X$, we will use mathematical induction to show that for $1 \leq k \leq n - 2$ it holds that $(k, k + 1) = X$. For $k = 1$, the base case, this follows trivially, since P_1 contains only one location that is not in the same column as $[n, 1]$, and this is $[1, 2]$, so indeed $(1, 2) = X$.

For the induction step assume that for $1 \leq m < k$ it holds that $(m, m + 1) = X$. It follows that for $1 \leq m \leq k$ there is an X in column m that is not on a location in P_k, since it is either in P_{m-1} (and $m \leq k$, so $P_{m-1} \neq P_k$) or at $[n, 1]$ which is in not in any P_a with $1 \leq a \leq n - 2$. Also, there is an X on row $k - 1$ that is not on a location in P_k. This excludes all elements in P_k except $(k, k+1)$ to be equal to X, so indeed it must be that $(k, k+1) = X$.

Call $(1, 1) = Y$. Since $[1, 1]$ is in the same column as $[1, n]$, and because $n \neq 1$ and $(1, n) = X$ we know $Y \neq X$. We will use mathematical induction to show that for $1 \leq k \leq n - 2$ it holds that $(k, k) = Y$. For $k = 1$, the base case, we just established this.

For the induction step assume that for $1 \leq m < k$ it holds that $(m, m) = Y$. It follows that for $1 \leq m < k$ there is a Y in column m that is not on a location in P_k, since it is in P_m (and $m < k$, so $P_m \neq P_k$). Also, there is a Y on row $k-1$ that is not on a location in P_k. This excludes all elements in P_k

except (k, k) and $(k, k+1)$ to be equal to Y, but since $(k, k+1) = X \neq Y$, it must be that $(k, k) = Y$.

It follows from the fact that for $k \leq n-2$ it holds that $(k, k) = Y$ that only $[n-1, n-1]$, $[n-1, n]$, $[n, n-1]$ and $[n, n]$ do not yet have a Y in their row or column. However, $(n-1, n) = X$, since $(k, k+1) = X$ for $k = n-2$. As a result, the only way to fill in two more Y's that are not in the same row or column, is to make $(n-1, n-1) = (n, n) = Y$.

The part of the grid that was not covered by the $n-2$ polyominoes defined before is the set $[n-2, n] \cup \bigcup_{x=2}^{n} [n-1, x] \cup \bigcup_{x=1}^{n} [n, x]$.

Now for any $1 \leq r \leq n-3$ we can define $P_{n-1} = [n-2, n] \cup [n, n] \cup \bigcup_{x=n-r}^{n} [n-1, x]$ and $P_n = \bigcup_{x=1}^{n-1} [n, x] \cup \bigcup_{x=2}^{x-r-1} [n-1, x]$. Both of these sets contain n elements, and they have an empty intersection with any polyomino defined other than themselves.

Since for any $1 \leq r \leq n-3$ the collection of polyominoes P_i with $1 \leq i \leq n$ is a valid polyomino cover and P_{n-1} contains both $[n-1, n-1]$ and $[n, n]$, while $(n-1, n-1) = (n, n) = Y$, we have proven that this construction gives a jigsaw sudoku puzzle without a valid solution. In fact, we constructed $n-3$ of them, and because none of these are equivalent under the symmetries of the square (one needs to only look at the role of P_1 and how its shape can not occur in column n (because this column is part of many polyominoes), not on row 1 (by P_2), and not on row n (by P_n)), this principle accounts for $8(n-3) = 8n-24$ of the impossible jigsaw sudoku polyomino covers for a given n in Table 4.

5.3 A second proof

Figure 24 serves to illustrate a second proof that for any n there exist a polyomino cover that can not be used to create a jigsaw sudoku. We will provide only a brief outline.

The green area on the right depicts some polyomino contained in the top three rows. The green area on the left contains only squares from the second and third row, the column of X_b and Y_b and the square on the fourth row immediately to the left of the column with X_b and Y_b.

Starting with placing Y_a we know the Y in the white polyomino can only go at the position marked Y_b. Consequently, the leftmost green area

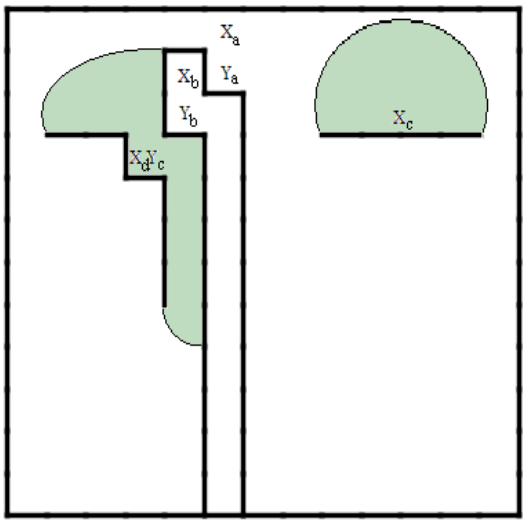

Figure 24: This image illustrates a second proof that for any n there exists a polyomino cover that can not be used to create a jigsaw sudoku.

can only have Y at the position marked Y_c.

The element above Y_a we mark X_a. Since this X and Y are in the same column, we know $X \neq Y$. Therefore there is an X at the position X_b. The green area on the right must have an X and it has to be on the third row. This means the leftmost green area can not have an X on the second or third row, nor can it be in the column of X_b. It's last refuge would be the square of Y_c, but since $X \neq Y$ it can not be there either. This concludes the proof provided that the remaining white area can be tiled with polyominoes of size n, which is no problem for $n \geq 5$ and the number of ways this can be done grows exponentially.

6 Acknowledgements

The author would like to thank Walter Kosters for his role as supervisor and Maarten Löffler for his help on searching for the smallest three dimensional cannibal.

References

[1] J. de Ruiter, Sudoku No. 346, PuzzlePicnic.com, Jul. 2007.

[2] Polyomino, Wikipedia,
http://en.wikipedia.org/wiki/Polyomino,
retrieved Jan. 2010.

[3] J. de Ruiter, Sudoku No. 1201, PuzzlePicnic.com, Jan. 2009.

[4] L. Alonso and R. Cerf, The Three Dimensional Polyominoes of Minimal Area, The Electronic Journal of Combinatorics, 3, R27, 1996.

[5] N. J. A. Sloane, The On-Line Encyclopedia of Integer Sequences,
http://www.research.att.com/~njas/sequences/

[6] D.E. Knuth, Dancing Links, Millennial Perspectives in Computer Science, 187–214, 2000.

[7] Dancing Links, Wikipedia,
http://en.wikipedia.org/wiki/Dancing_Links,
retrieved Feb. 2010.

[8] bbi5291, Computer Science Canada,
http://compsci.ca/v3/viewtopic.php?t=21164,
retrieved Mar. 2010

[9] Derangement, Wikipedia,
http://en.wikipedia.org/wiki/Derangement,
retrieved Feb. 2010.

[10] B. Harris, An N-1 Hint NxN Du-Sum-Oh Exists for Any N,
http://www.bumblebeagle.org/dusumoh/proof/, 2006.

[11] J.A. Bate and G.H.J. van Rees, Minimal and Near-Minimal Critical Sets in Back-Circulant Latin Squares, Australasian J. Combinatorics 27, 47–61, 2003.

[12] J. de Ruiter, Sudoku No. 780, PuzzlePicnic.com, Apr. 2008.

[13] J.A. Bate and G.H.J. van Rees, The Size of the Smallest Strong Critical Set in a Latin Square, Ars Combin. 53, 73–83, 1999.