

Counting Nonomino Tilings and Other Things of that ilk.

Bob Harris, March/13/2010, for G4G9.

Some time during 2009 I received a question about the Grand Time Sudoku that appeared in [1]. The correspondent asked if this puzzle was the result of an exhaustive search over all possible 9×9 jigsaw sudoku boards. The answer was that, while it was the result of a search, it was far from exhaustive. The query got me thinking, though; just how many possible boards are there? Was this answer known?

More formally, I wanted to know how many ways there are to tile a 9×9 board with nine nonominoes. I allow the same nonomino to be used more than once, and I consider symmetrically equivalent tilings to be distinct (that is, if two tilings are equivalent by, say, rotation, I count both tilings). From this angle the problem is a generalization of the domino tiling problems discussed in [2].

I was unable to determine whether the 9×9 problem had previously been solved. After computing counts for various small piece sizes and small boards using existing programs, I found several related sequences in OEIS [3] — counts for tiling with trominoes (A134438¹), for tiling with a combination of monominoes and dominoes (A030186), and for tiling with 9 regions of equal area (A167265). The closest I came to finding a solution was A167255, the number of ways to partition an $N \times 7$ grid into 7 connected equal-area regions, which gave the answer for a 7×7 board. A search of the literature also failed to yield an answer for the 9×9 .

I also suggested the problem at the yahoo polyforms group (a group devoted to polyomino tiling problems). Initial estimates were that this might be computable in thousands of years; after some experiments the estimate was reduced to 80 days for 8×8 , 100 years for 9×9 . But these estimates were based on a placement/search algorithm, a depth first search of all possible placements of pieces. I felt this was doing more work than necessary, since I didn't need to know the actual tilings, just a count.

The other method I'd seen for tiling counting was to derive (and evaluate) a recurrence relation (as in [4]). For all but the smallest problems the recurrence relation is beyond human capabilities. My approach would be to automate this; similar to what Michael Reid had done for congruent polyomino tilings in [5]. The puzzle was how to compute this in a reasonable time. In this context, a reasonable time was something less than a day with a typical desktop or laptop computer that an average person would have. In the end, the programming solution required a surprising number of items from the algorithmic bag of tricks, mostly involving memory utilization, none of which I will bother to mention in this manuscript².

Before I describe how I solved it, to relieve the suspense, there are about 700 trillion³ 9×9 tilings. Whether an exhaustive search for jigsaw sudoku is possible is hard to say. Reducing by symmetry would leave us about 100 trillion tilings to search. If we could generate and process three billion tilings a second it would "only" take 9 hours for the search. With large-scale parallelization this could be within reach with today's supercomputer technology. With the typical desktop or laptop computer I think it is still out of range. Note that while my solution does not *generate* the tilings (it only counts how many there are), I do sketch a backtracking technique that could potentially perform generation.

¹ Strings of this form will indicate sequences in OEIS. A list of relevant OEIS sequences appears in the appendix.

² The program is freely available, though, and the author will be more than happy to discuss it.

³ A more exact answer appears later in this manuscript.

Computing by Recurrence

Consider the four tilings in Figure 1. Given a cursory glance, these tilings apparently have little in common. However, when sliced between the third and fourth rows they all exhibit the same *slice state* in their third row. The bottom three rows of each tiling are shown in Figure 2, after removing any region that does not touch the slice line. The four partial tilings are equivalent in that (a) they divide the top row into four partial regions with 2, 2, 4 and 1 squares on the slice (scanning left to right), and (b) these partial regions have areas 4, 3, 8 and 3 below the slice. This slice state is diagrammatically simplified in Figure 3(a). Nothing else that occurs below the slice will affect how many ways this slice can be extended into a tiling of the full square.

There are two further points worth making regarding slice states. First, note that the partial tiling in 2(d) must be flipped horizontally to match the criteria for this state. We consider states that are mirror images of each other to be equivalent for counting purposes. Second, the sum of the areas of the partial regions is necessarily a multiple of N . This fact reduces the number of possible states by a factor of about N .

The counting program works by evaluating the recurrence relation between slice states on successive rows. The possible states for a slice above row one are counted, each of these is expanded to a set of achievable slice states above the second row, the second row states are counted, and so on. At the 9th row we discard any states with partial regions (regions with area $\neq 9$). The counts for the remaining states give the number of tilings for the full square.

The method is best described by example. Since the number of slice states for the nonomino problem is too huge⁴, I'll describe the problem of tiling $3 \times N$ boards with trominoes. The program identifies seven different slice states for the tromino problem, which are shown in Figure 4.⁵ Figure 5 shows all the steps from one row's slice state to the next. The figure leads directly to a recurrence relation for computing the number of tilings of a $3 \times N$ board. With the count for the number of ways to reach slice state S after N rows being $S(N)$, we have equation 1.⁶ Computing this for $1 \leq N \leq 10$ gives us the counts shown in table 1.

$$\begin{array}{ll}
 A(N) = A(N-1) + C(N-1) + D(N-1) + E(N-1) + G(N-1) & A(0) = 1 \\
 B(N) = C(N-1) & B(0) = 0 \\
 C(N) = D(N-1) & C(0) = 0 \\
 D(N) = B(N-1) + G(N-1) & D(0) = 0 \\
 E(N) = F(N-1) & E(0) = 0 \\
 F(N) = A(N-1) & F(0) = 0 \\
 G(N) = 2A(N-1) & G(0) = 0
 \end{array} \quad (\text{eq. 1})$$

⁴ The margin truly is too small to contain all the necessary states.

⁵ States A, A', and A'' are treated as a single state.

⁶ Human ingenuity can reduce this recurrence relation substantially. However, since the program makes no such simplification, I show the full recurrence.

For larger pieces sizes the number of slice states explodes—the rate of growth is a little larger than N^{N-1} . In addition to having more columns and more possible region sizes, beginning with pentominoes a slice may contain non-contiguous regions. An example of this occurs in the tiling of Figure 1(c), with the slice at the top of the fifth row — the slice state is shown in Figure 3(b). Table 2 (second column) shows the number of slice states the program used for the each piece size. The number of slice states shown is the maximum number of states reachable from the starting condition. Some states are not reachable within the first N rows, so the actual number of states involved in solving the $N \times N$ board was slightly lower⁷.

To determine the recurrence steps from a particular state, the program considers the 2^{2N-1} possible present/absent configurations of the segments for the next row (N horizontal segments and $N-1$ vertical segments). It eliminates configurations that would merge neighboring regions, allow regions that are larger than N , or trap regions that are smaller than N . Most configurations are rejected⁸. For the 9×9 a state has 16.3 acceptable configurations on average. Early versions of the program cached the list of next states for a given state, but for the 9×9 problem this required too much memory and was abandoned in favor of re-computation.

Results

Table 2 (third column) shows the counting results for tiling an $N \times N$ square with N N -ominoes. This appears to grow at a rate of $N^{Q(N)}$, where $Q(N)$ is a quadratic⁹. Runtime to compute the 9×9 is under 4 hours on a 2.4GHz Intel Mac. After that, about 52 minutes is needed to compute each subsequent row. Memory was a key factor in being able to solve the 9×9 , as about 1.6G bytes is needed to maintain current-row and next-row counts for the roughly 45 million states.

Table three shows counts for tiling an $N \times M$ rectangle with N -ominoes, for up to ten rows and piece sizes from dominoes to nonominoes.

Related Problems

Mixed Piece Sizes

The above discussion assumes that all pieces are of the same size. However, the program can easily handle mixed piece sizes. One change is that the previous criteria to “eliminate configurations with regions larger than N , or trapping regions smaller than N ”, is restated as “eliminate configurations with regions too large, or that trap regions that are too small.” Thus we can set a range of piece sizes P_{\min} to P_{\max} . The state space is also affected. First, if the largest piece could envelope the smallest piece, the states have to contain additional information to track region adjacency¹⁰. Second, the sum of the areas of the partial regions will no longer be a multiple of N . Both these effects serve to drastically increase the number of reachable states.

⁷ For example, for 9×9 only 49,711,158 states are used (99.98% of the possible states). The full set of states is not reached until 9×11 .

⁸ In reality, the program performs a breadth first search on each row, scanning the row from left to right, and considering the segments two at a time, rather than considering configurations of all $2N-1$ segments at once.

⁹ A good fit is achieved with $Q(N) = (69N^2 + 73N - 16) / 400$.

¹⁰ Technically this would also be true for fixed piece sizes of 23 or larger.

Setting P_{\min} to 1 and P_{\max} to NM yields the number of ways to partition an $N \times M$ rectangle into regions of any size. This problem appeared as an internet programming contest in 2007 [6]. Table 4 shows the partition counts for up to a 5×10 rectangle¹¹. Tables 5 and 6 show counts when partitions are required to have size at least 2 or 3. The first column of table 6 is identical to A000930, revealing that the number of ways to tile $N \times 3$ with straight trominoes is the same as the number of ways to tile $(N+3) \times 1$ with planks of length 3 or longer.

Generating Tilings, or Randomly Sampling Tilings

One of the disadvantages of this method, compared to the piece placement method, is that the latter can report each tiling. With enough memory, the recurrence method could keep track of the count for each state at each row and by using backtracking a depth-first scan over all the tilings could be performed. Or, it could use backtracking to map a random integer into a tiling, effectively providing uniform random sampling from the tilings. There seems to be some interest in uniform sampling from tilings. See for example [7], which sketches a scheme for sampling from tilings with T-tetrominoes.

For example, for tiling 5×100 with pentominoes, the “forward” pass would generate and save the recurrence (about 750 states and 3200 recurrence pairs could easily be encoded in 13K) and counts for each row (750 counts at 40 bytes each¹², or another 30K per row). Total memory should be no more than 3M bytes, though this can certainly be reduced by clever storage.

For the targeted 9×9 problem we have about 50 million states and 800 million recurrence pairs. This could be encoded in less than 11G bytes (7.2G for the recurrence and 3.6G for the counts).

Availability

Two programs implementing this method, one written in python, the other in C, are freely available at www.bumblebeagle.org/polyominoes/tilingcounting. The author can be contacted at [bobplastic @ wrapperbumblebeagle.org](mailto:bobplastic@wrapperbumblebeagle.org) (but you must carefully remove the plastic wrapper).

Epilog

I began my work on this project in January 2010 and first successfully computed the count for the 8×8 on February 1st. At that time my program needed 300 million states to solve the 9×9 , which outstripped the memory on my machine. Patrick Hamlyn (of the yahoo polyforms group) graciously offered to use his computer to run my program, and the 9×9 was first cracked on February 11th.

While writing this manuscript, I discovered A172477, which had been added to OEIS in the interim. This led me to Johan de Ruiter, who had been working on the same problem and had cracked the 8×8 in 2008, but who had abandoned his attempt at the 9×9 [8]. This provided corroboration for my 8×8 result.

Several of the OEIS sequences listed in the appendix are due to Ron Hardin and are also fairly recent, having been posted in late 2009. Apparently he was able to crack the 7×7 and got close to 8×8 .

¹¹ Extending A110476, and computed in less than 15 seconds.

¹² The count of 5×100 tilings is $\approx 8.5 \times 10^{87}$, requiring 37 bytes.

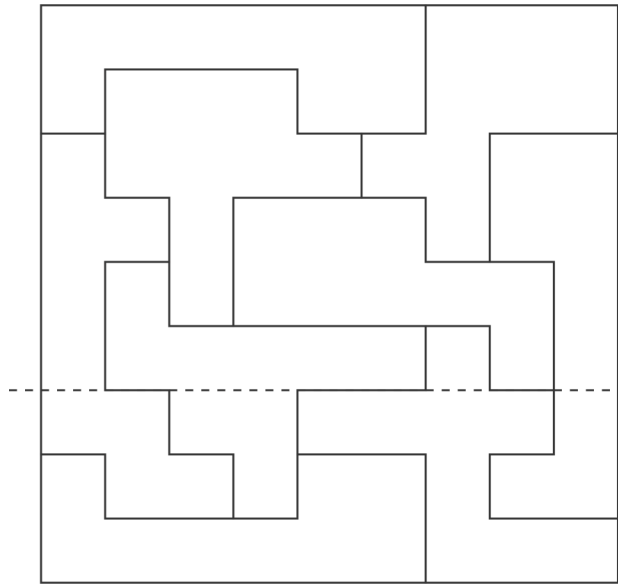
Appendix

Relevant sequences found in OEIS [3]:

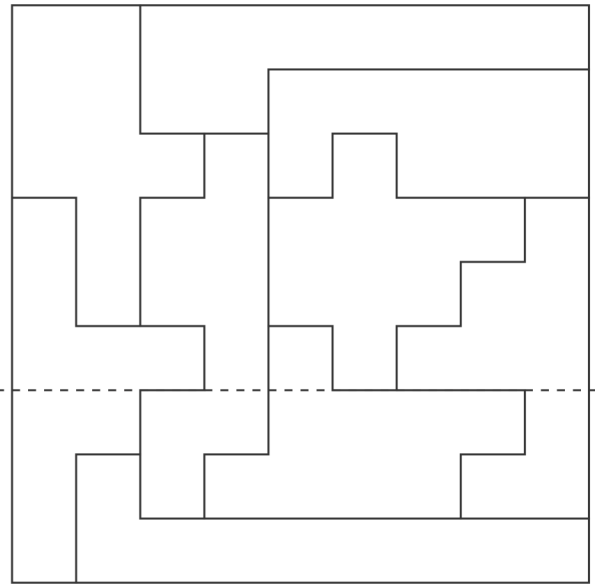
- A000930 $a(0) = a(1) = a(2) = 1$; thereafter $a(n) = a(n-1) + a(n-3)$. (Number of tilings of an $N \times 3$ rectangle with straight trominoes.)
- A030186 $a(n) = 3a(n-1) + a(n-2) - a(n-3)$; $a(0) = 1$, $a(1) = 2$, $a(2) = 7$. (Number of partitions of an $n \times 2$ rectangle into monominoes and dominoes.)
- A110476 Number of partitions of an $M \times N$ rectangle.
- A134438 Number of tilings of a $3 \times N$ rectangle with “triminoes.”
- A167243 Number of ways to partition an $N \times 3$ grid into 3 connected equal-area regions.
- A167248 Number of ways to partition an $N \times 4$ grid into 4 connected equal-area regions.
- A167251 Number of ways to partition an $N \times 5$ grid into 5 connected equal-area regions.
- A167254 Number of ways to partition an $N \times 6$ grid into 6 connected equal-area regions.
- A167255 Number of ways to partition an $N \times 7$ grid into 7 connected equal-area regions.
- A167258 Number of ways to partition an $N \times 8$ grid into 8 connected equal-area regions.
- A167265 Number of ways to partition an $N \times M$ grid into 9 connected equal-area regions.
- A172477 Number of ways to dissect an $N \times N$ square into polyominoes of size N .

References

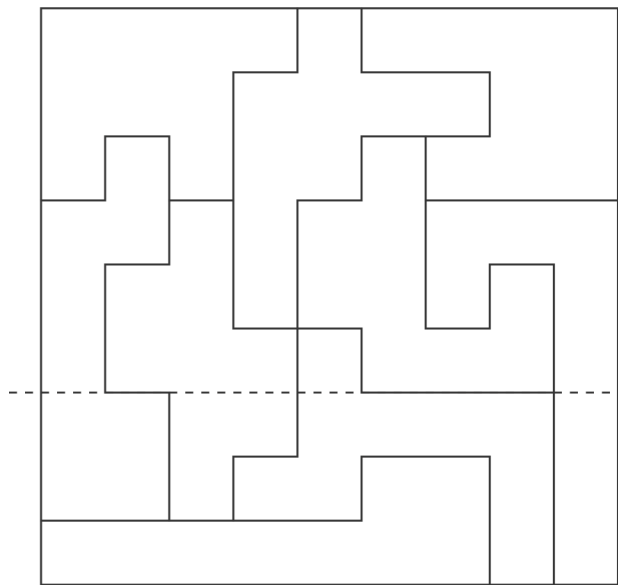
- [1] Harris RS. “The Grand Time Sudoku and the Law of Leftovers”, in “Mathematical Wizardry for a Gardner”, Pegg Jr E, Schoen AH, Rodgers T, eds. pp. 55-57 (2009).
- [2] Kotani S. “Y2K Problem of Dominoes and Tatami Carpeting”, in “Puzzler’s Tribute, A Feast for the Minds”, Wolfe D and Rodgers T, eds. pp. 413-420 (2002).
- [3] Sloane NJA. The On-Line Encyclopedia of Integer Sequences, <http://www.research.att.com/~njas/sequences>.
- [4] Katz M and Stenson C. “Tiling a $(2 \times n)$ -Board with Squares and Dominoes”, Journal of Integer Sequences, no. 12 (2009).
- [5] Reid M. “Tiling Rectangles and Half Strips with Congruent Polyominoes”, Journal of Combinatorial Theory, series A 80, no. 1, pp. 106-123 (1997).
- [6] Internet Problem Solving Contest, IPSC 2007, Problem D - Delicious Cake. <http://ipsc.ksp.sk/contests/ipsc2007/real/problems/d.php> (2007).
- [7] Korn M and Pak I. “Tilings of rectangles with T-tetrominoes”, Theor. Comp. Sci. no. 319 pp. 3-27 (2004).
- [8] de Ruiter J. “On Jigsaw Sudoku Puzzles and Related Topics” Bachelor Thesis, Leiden Institute of Advanced Computer Science (2010).



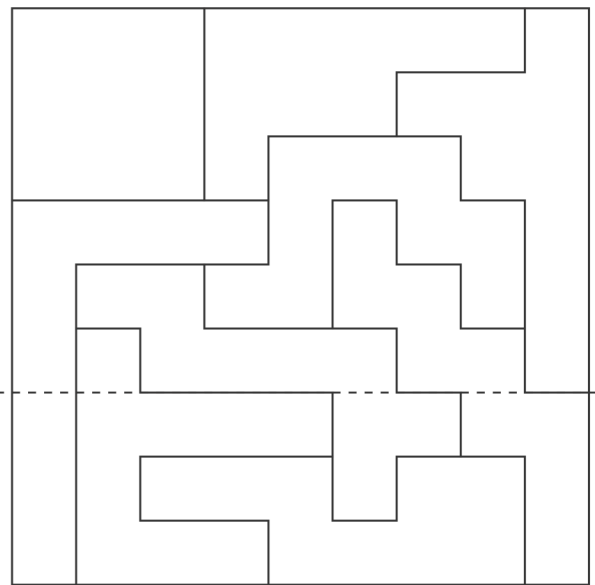
(a)



(b)



(c)



(d)

Figure 1. Example tilings of a 9x9 square.

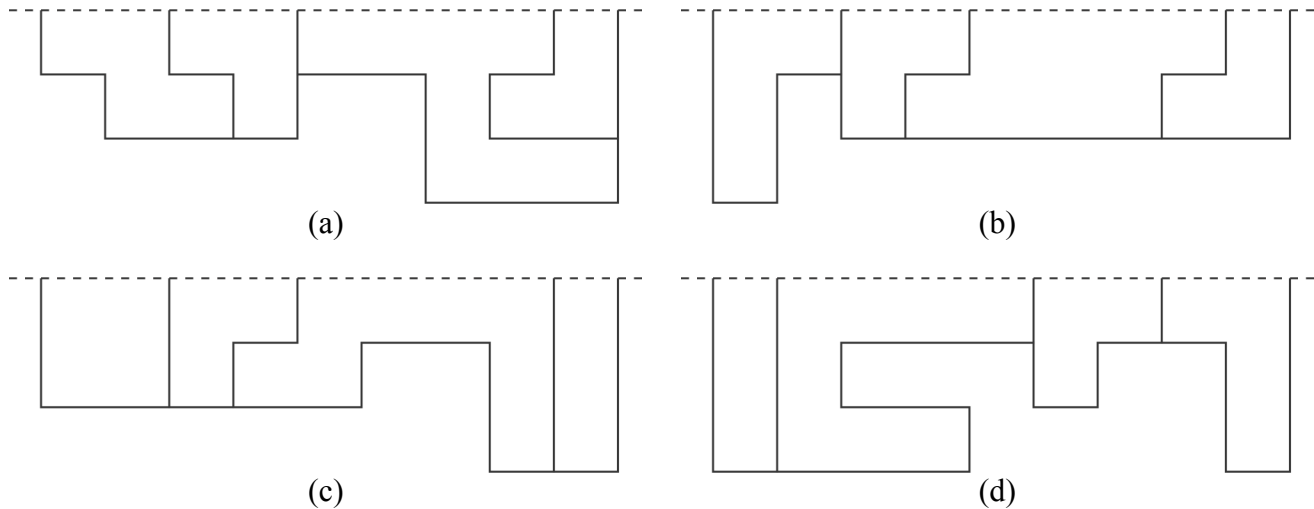


Figure 2. Three-row partial tilings.

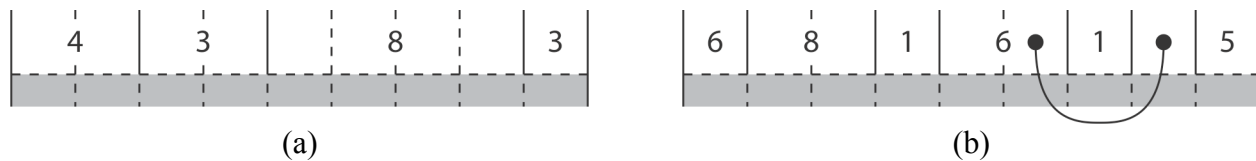


Figure 3. Two slice states.

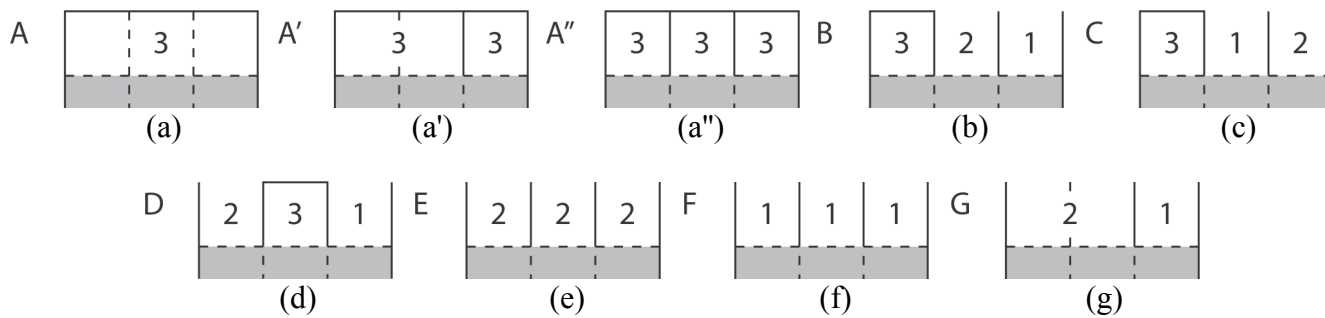


Figure 4. Slice states for $N \times 3$ tromino tiling.

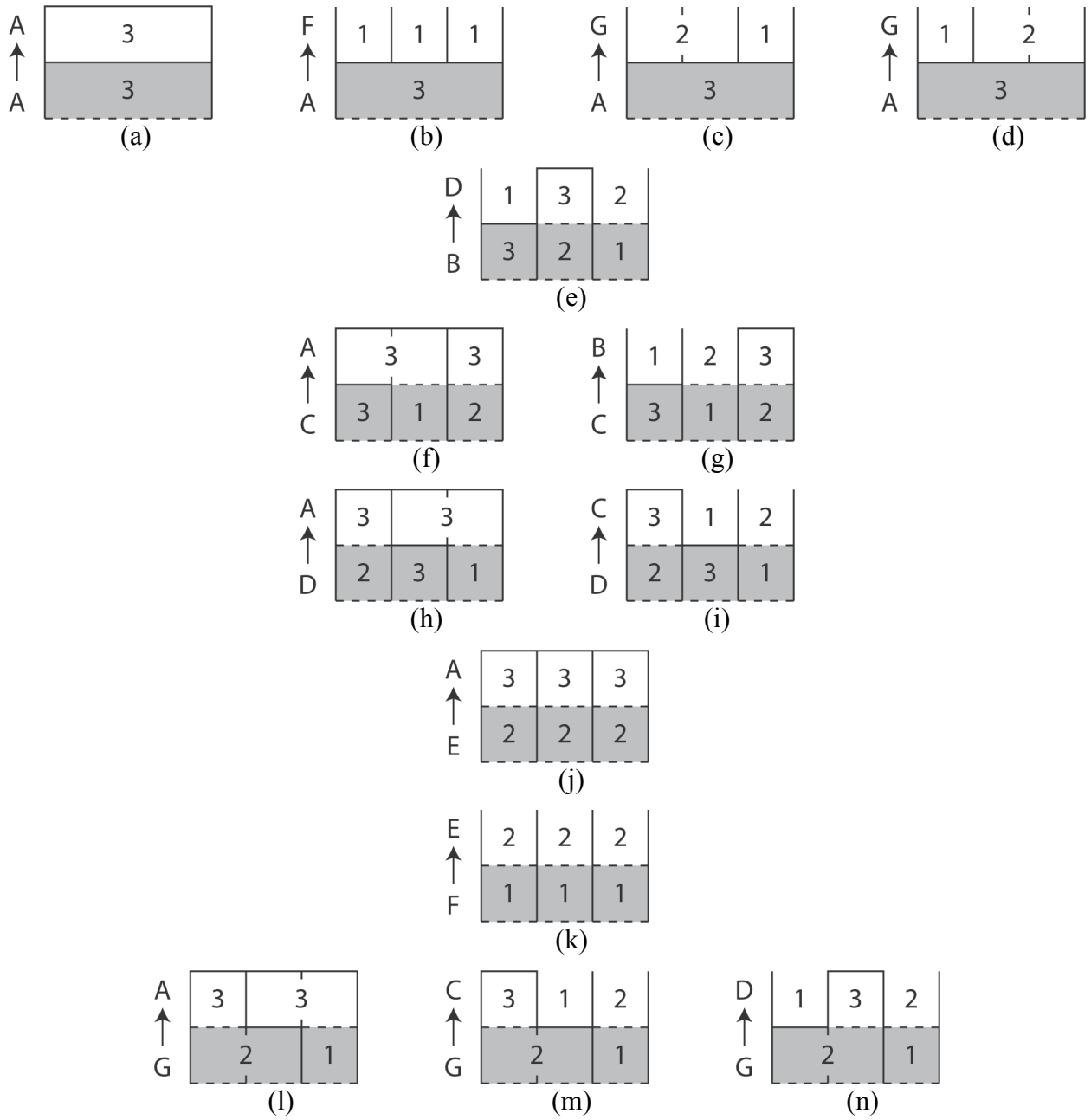


Figure 5. Slice state recurrence steps for $N \times 3$ tromino tiling.

Board $3 \times N$	3×1	3×2	3×3	3×4	3×5	3×6	3×7	3×8	3×9	3×10
Tilings $A(N)$	1	3	10	23	62	170	441	1,173	3,127	8,266

Table 1. Tromino tiling counts (A134438).

N	Slice states for N columns	$N \times N$ tilings
2	2	2
3	7	10
4	51	117
5	508	4,006
6	6,758	451,206
7	108,764	158,753,814
8	2,055,081	187,497,290,034
9	44,362,920	706,152,947,468,301

Table 2. Counts for tiling an $N \times N$ square with N N -ominoes (A172477).

$N \rightarrow$	2	3	4	5	6	7
$N \times 2$	2	3	4	5	6	7
$N \times 3$	3	10	23	56	132	259
$N \times 4$	5	23	117	501	2,369	9,525
$N \times 5$	8	62	454	4,006	33,344	270,827
$N \times 6$	13	170	2,003	27,950	451,206	6,633,399
$N \times 7$	21	441	9,157	214,689	5,850,115	158,753,814
$N \times 8$	34	1,173	40,899	1,696,781	81,459,922	3,825,111,851
$N \times 9$	55	3,127	179,399	13,205,354	1,144,259,389	96,608,374,284
$N \times 10$	89	8,266	796,558	101,698,212	15,946,621,499	2,446,223,788,303

Table 3(a). Counts for tiling an $N \times M$ rectangle with M N -ominoes.

$N \rightarrow$	8	9
$N \times 2$	8	9
$N \times 3$	546	1,095
$N \times 4$	39,731	145,415
$N \times 5$	2,152,050	15,661,597
$N \times 6$	99,697,633	1,418,159,011
$N \times 7$	4,292,655,082	112,976,454,947
$N \times 8$	187,497,290,034	8,812,020,831,683
$N \times 9$	8,378,760,802,160	706,152,947,468,301
$N \times 10$	385,296,986,628,990	58,015,563,977,931,125

Table 3(b). Counts for tiling an $N \times M$ rectangle with M N -ominoes.

N/M	1	2	3	4	5
1	1	2	4	8	16
2	2	12	74	456	2,810
3	4	74	1,434	27,780	538,150
4	8	456	27,780	1,691,690	103,015,508
5	16	2,810	538,150	103,015,508	19,719,299,768
6	32	17,316	10,424,872	6,273,056,950	3,774,627,281,350
7	64	106,706	201,947,094	381,992,581,548	722,529,289,997,430
8	128	657,552	3,912,050,356	23,261,112,447,444	138,304,601,846,374,768
9	256	4,052,018	75,782,907,270	1,416,465,537,909,008	26,473,890,580,295,999,056
10	512	24,969,660	1,468,040,672,696	86,254,456,382,365,422	5,067,560,038,024,745,010,778

Table 4. Counts for partitioning an $N \times M$ rectangle into regions of any size (A110476).

N/M	1	2	3	4	5
1	0	1	1	2	3
2	1	3	13	53	217
3	1	13	147	1,659	18,545
4	2	53	1,659	50,276	1,524,115
5	3	217	18,545	1,524,115	124,826,337
6	5	891	207,837	46,201,303	10,237,173,673
7	8	3,657	2,328,129	1,400,627,890	839,355,395,473
8	13	15,009	26,081,019	42,459,830,147	68,821,250,748,463
9	21	61,601	292,169,417	1,287,166,237,267	5,642,825,358,513,511
10	34	252,827	3,273,004,955	39,020,336,218,258	462,669,635,038,911,681

Table 5. Counts for partitioning an $N \times M$ rectangle into regions of at least size 2.

N/M	1	2	3	4	5
1	0	0	1	1	1
2	0	1	4	13	44
3	1	4	43	373	3,174
4	1	13	373	8,210	176,795
5	1	44	3,174	176,795	9,616,087
6	2	143	27,271	3,855,662	530,842,817
7	3	480	234,353	84,158,827	29,331,810,977
8	4	1,601	2,012,822	1,834,874,258	1,618,678,503,422
9	6	5,338	17,289,627	40,007,220,226	89,332,805,336,907
10	9	17,793	148,513,953	872,381,043,912	4,930,582,959,283,185

Table 6. Counts for partitioning an $N \times M$ rectangle into regions of at least size 3.