Discrete Mathematics 36 (1981) 191–203
North-Holland Publishing Company

# COUNTING POLYOMINOES: YET ANOTHER ATTACK

D. Hugh REDELMEIER
*Department of Computer Science, University of Toronto, Toronto, Ontario, M5S 1A7 Canada*

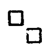
Received 18 September 1979
Revised 12 August 1980

A polyomino is a connected collection of squares on an unbounded chessboard. There is no known formula yielding the number of distinct polyominoes of a given number of squares. A polyomino enumeration method, faster than any previous, is presented. This method includes the calculation of the number of symmetric polyominoes. All polyominoes containing up to 24 squares have been enumerated (using ten months of computer time). Previously, only polyominoes up to size 18 were enumerated.

## 1. What is a polyomino?

A *d*omino is a pair of equal sized squares touching along a complete edge (we ignore the spots). Generalizing, a *pol*yomino is a collection of equal-sized squares in a plane, touching each other along complete edges. We call these squares *cells*. Here are a few examples:

Every cell need not touch every other one, but a polyomino must be connected. Thus, the pair of cells

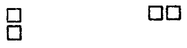


do not constitute a polyomino since they are not connected.

Formally, a *polyomino* is a connected graph in which each node (or *cell*) is identified with a point in the Cartesian lattice, and edges of the graph join nodes that are separated by a unit distance. The *size* of a polyomino is the number cells it contains.

The question that this paper is concerned with is "How many polyominoes of size $p$ are there?" The answer depends on how we distinguish polyominoes. There are two sets of distinguishing rules commonly used, and for each set there is a name for the polyominoes.

*Free* polyominoes are considered distinct if they have different shapes. Their orientation and location in the plane is of no importance. For example, the two polyominoes

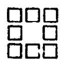


are the same free polyomino since they differ only in orientation. We use free($p$) to denote the number of free polyominoes of size $p$.

*Fixed* polyominoes are considered distinct if they have different shapes or orientations. Thus the two polyominoes above are different fixed polyominoes. We use fixed($p$) to denote the number of fixed polyominoes of size $p$.

## 2. Related work

The most general discussion of polyominoes is by Golomb [1], however the number of polyominoes is only briefly discussed. Unlike most later authors (including us) Golomb does not allow holes in polyominoes. Thus he would not accept the following as a polyomino.



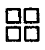
Read [2] derived several theoretical results about the number of polyominoes. He presented a method for deriving generating functions to calculate the number of polyominoes, but these become intractable very quickly. He calculated free($p$) for $p$ up to 10, but his value for size 10 was incorrect.

Klarner [3] found bounds for free($p$) and fixed($p$); the upper bound was improved by Klarner and Rivest [4]. The limits of the $p$th roots of free($p$) and fixed($p$), as $p$ approaches infinity, were shown to be equal and between 3.72 and 4.65. Obviously then, free($p$) and fixed($p$) are exponential in $p$.

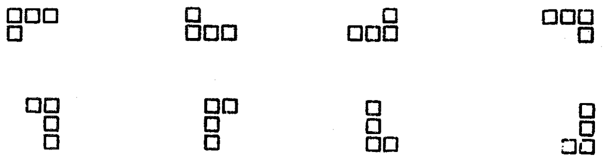
Lunnon [5] has made the most successful previous enumeration. He computed the number of free, fixed, and symmetric polyominoes up to size 18. We believe his results for size 17 are incorrect. Our work is most closely related to Lunnon's.

## 3. Symmetries of polyominoes

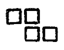
The key to the difference between fixed and free polyominoes is the symmetries of polyominoes. Every free polyomino corresponds to one, two, four, or eight fixed polyominoes, depending on its symmetry. For example, a very symmetric free polyomino, such as



corresponds to only one fixed polyomino. An asymmetric free polyomino, on the other hand, can be rotated and reflected to yield eight fixed ones, as in this example:



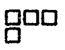
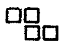
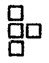
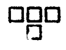
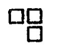
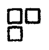
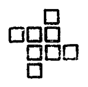
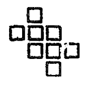
A polyomino is said to have a certain *symmetry* if it is invariant under the transformation(s) associated with that symmetry. Since, for example, the polyomino



is unchanged when rotated by $\pi$ radians, it is said to be rotationally symmetric. Symmetries of free and fixed polyominoes are similar, but only fixed polyomino symmetries involve orientation (for example, reflection in the *horizontal* axis).

Table 1 catalogues the symmetries of polyominoes. The parenthetical note in a transformation (the last column) orients it for fixed symmetries; without this orientation, the transformation is that of a free symmetry. Two pairs of transformations differ only in orientation, and therefore each pair represents one free symmetry. The index of a symmetry is the number of fixed polyominoes corres-

Table 1

| Free | Fixed | Index | Example | Transformation(s) |
|---|---|---|---|---|
| none | N | 8 |  | none |
| rot | R | 4 |  | rotation by $\pi$ radians |
| axis | H | 4 |  | reflection in (horizontal) axis |
| axis | V | 4 |  | reflection in (vertical) axis |
| diag | A | 4 |  | reflection in (ascending) diagonal |
| diag | D | 4 |  | reflection in (descending) diagonal |
| rot 2 | R2 | 2 |  | rotation by $\frac{1}{2}\pi$ radians |
| axis 2 | HVR | 2 | | reflection in either axis, or rotation by $\pi$ radians |
| diag 2 | ADR | 2 |  | reflection in either diagonal, or rotation by $\pi$ radians |
| all | HVADR2 | 1 | | all of the above |

ponding to each free polyomino with that symmetry. The columns labelled 'Free' and 'Fixed' give our names for the free and fixed symmetries, respectively.

We call each symmetry with index 4 *simple* because it has one transformation associated with it. Each symmetry with index less than 4 is *composite* since it has several transformations associated with it. Note that the transformation of R2 subsumes rotation by $\pi$ radians. Each composite symmetry includes simple symmetries in the sense that any polyomino with a composite symmetry will also have simple symmetry: one simple symmetry for each of the composite symmetry's transformations. For example, HVR symmetry includes H, V, and R symmetries. Not all combinations of simple symmetries exist as composite symmetries because some combinations imply further symmetry. Thus H and V symmetry together imply R symmetry.

We shall use the name of a symmetry to denote the function that maps a polyomino size into the number of polyominoes of that size having *only* that symmetry. Thus N($p$) is the number of fixed polyominoes of size $p$ with no symmetry. We shall also put a prime after the symmetry name to denote the mapping yielding the number of polyominoes with *at least* that symmetry. Thus

$$\mathrm{H}'(p) = \mathrm{H}(p) + \mathrm{HVR}(p) + \mathrm{HVADR2}(p)$$

N′($p$) and fixed($p$) are obviously equivalent.

Most polyominoes have no symmetry, since requiring a symmetry reduces the "degree of freedom" in constructing a polyomino. With any simple symmetry, the position of roughly half the cells almost completely specifies where the remaining cells must be placed. Since fixed($p$) is exponential in $p$, the number of simply symmetric polyominoes of size $p$ is roughly proportional to $\sqrt{\mathrm{fixed}(p)}$. Similarly, there are many fewer polyominoes with composite symmetries than with simple symmetry. Since most polyominoes are asymmetric, fixed($p$) is close to eight times free($p$), for large enough $p$.

## 4. How many polyominoes are there?

Although it is known that both free($p$) and fixed($p$) are exponential in $p$ there is no known formula for either. To calculate values for them, we must fall back on exhaustive enumeration.

Enumerating polyominoes is very expensive. The cost is directly proportional to the number enumerated, and, in the range we are concerned with, each unit increase in $p$ increases the numbers of polyominoes by a factor of almost four. The most successful enumeration to date [5] used 175 hours of computer time. Clearly, to get much farther, we must be much more efficient.

For a given $p$, we would like to generate and count every free polyomino of size $p$ exactly once. We do not know how to do this without generating a larger set and rejecting unacceptable members, expending effort in generating useless configurations, and in testing each configuration. In fact, this larger set is at least the fixed polyominoes.

We do know how to generate fixed polyominoes without generating useless configurations. It turns out that this generation is quite efficient, largely because no configuration need be rejected and therefore no testing is needed. In fact, we can compute fixed($p$) faster than free($p$) even though there are almost eight times as many polyominoes to count.

We have also developed separate methods to generate polyominoes that are constrained to have at least a given simple fixed symmetry. These methods are almost as fast, per polyomino counted, as the fixed polyomino generator. Thus we can calculate the primed simple symmetry functions relatively cheaply. We can also piggyback on these the primed composite symmetry function computations.

There is a well known way to count cattle in a herd: count the number of legs and divide by four. We apply this technique in calculating free($p$). We first enumerate fixed($p$). We then separately enumerate the primed symmetry functions, and derive from them their unprimed counterparts. Finally, we derive free($p$) from these figures: free($p$) is the sum, for each fixed polyomino, of the inverse of the index of the polyomino's symmetry. Thus, an asymmetric fixed polyomino counts as one-eigth of a free polyomino, whereas an HVADR2 fixed polyomino counts as one free polyomino.

Since the number of symmetric polyominoes is proportional to $\sqrt{\text{fixed}(p)}$, and since the cost of our enumeration is proportional to the number of items enumerated, the dominant computation will be the enumeration of fixed($p$).

## 5. An Algorithm to enumerate fixed polyominoes

To ensure fixed polyominoes are counted exactly once, we define a canonical form for them. In this form there is a cell at the origin of the Cartesian lattice, no cells below the $x$-axis, and no cells to the left of the origin on the axis. This forces the left-most cell of the bottom row of the polyomino to be at the origin.

The algorithm calculates fixed($p$) for all $p$ up to a specified limit $P$, using a depth-first traversal of a "family tree" of all polyominoes. 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 (however their offspring might). Equivalently, no younger brother or younger brother's offspring will be allowed to contain the cell. All of the child's offspring inherit the cell so that the child and its offspring are all different from the child's younger brothers and the younger brothers' offspring. A child is different from all its ancestors. Since being different is symmetric, it can be seen that every node in the tree must be different from every other.

The following routine is given a polyomino (the parent) and a set of untried points. The *untried set* contains all points that are adjacent to the parent and have not been used by the ancestors (including the parent) or the ancestors' older brothers. The following steps are repeated until the untried set is exhausted. Each iteration generates a child of the parent. Each recursion generates all the offspring of the current child.

1. Remove an arbitrary element from the untried set.
2. Place a cell at this point.
3. Count this new polyomino.
4. If the size is less than $P$:
   (a) Add *new* neighbours to the untried set.
   (b) Call this algorithm recursively with the new parent being the current polyomino, and the new untried set being a copy of the current one.
   (c) Remove the new neighbours from the untried set.
5. Remove newest cell.

The algorithm is started with the parent being the empty polyomino, and the untried set containing only the origin.

At any one time, each point in the field can be in one of four states:

*Border*: points below the $x$-axis and points left of the $y$-axis on the $x$-axis.

*Occupied*: points occupied by cells of the polyomino.

*Reachable*: unoccupied non-border points adjacent to cells of the polyomino.

*Free*: points that are none of the above, and therefore candidates for becoming Reachable and then Occupied.

The only information actually needed by the algorithm is whether a point is Free or not. This information allows Step 4(a) to decide if a neighbour is new or not. The information is updated in Step 4(a) when the new points are added to the untried set, and then restored in Step 4(c) when they are removed. Note that the information is not changed in Step 1.

The untried set contains those Reachable points that have not been tried by the current invocation of the algorithm or its recursive ancestors. While a point is in the untried set at some level in the recursion, it cannot be added to the untried set of another level. Otherwise a point could become a new neighbour while it is an old one. A point may, however, be in several untried sets at once due to inheritance. An invocation of the algorithm will, *itself*, count each polyomino containing the parent and a cell at a point in the untried set (the parent's children). An invocation will, *by itself and using recursion*, count each polyomino containing the parent and cells that are at points in the untried set or points currently Free (the parent's offspring).

The operations performed on the untried set are:

- removing an arbitrary element (Step 1),
- adding new elements (Step 4(a)),
- copying and passing as a parameter (Step 4(b)),
- removing new elements (Step 4(c)).

All operations can fit into a *stack* discipline where elements are removed in the reverse of the order they were added. The arbitrary element chosen in Step 1 would then be the newest element.

The *successor* of an element in the untried set is the next-newest element. Note that, due to the stack discipline, the successor of an element never changes. Even though a point can be an element in several untried sets, its successor in each case will be the same point: a point's successor is determined when the point becomes a new neighbour, and is not changed by inheritance.

A natural way to implement the untried set is, for each point in the field, to have a place to name its successor in all current untried sets. This allows an untried set to be represented by the name of its first (newest) element, with each element naming its successor—a *linked list*. All the operations used are then quite efficient:

- Removing an element is done by considering the successor of the current first element to be the new first element.

- Adding a new point is done by setting the successor of the new point to be the current first element, and then considering the new point to be the first element.

- Passing the list as a parameter is done by passing the name of the first element. Thus the whole set need not be copied.

Both Free information and successor information are naturally represented as matrices, with one element for each point in the field that can be used. These matrices represent points with $y$ co-ordinates between $-p+1$ and $p-1$, and $x$ coordinates between $-1$ (for the border) and $p-1$. Slightly more than half the points within these bounds can never be used, but this causes no problems. Locations of points are then naturally represented as subscripts for these matrices.

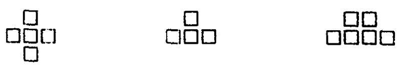
All polyominoes generated will be in canonical form. All will have a cell at the origin because that is the only point in the initial untried set. No points will be placed below the $x$-axis or on it to the left of the origin because of the definition of Free for determining new neighbours.

## 6. Enumerating symmetric fixed polyominoes

In this section, we describe how to compute the number of symmetric fixed polyominoes of each kind: the "primed" functions. We shall see that some of the functions can be derived from others, but some require enumeration. All computations involve modified versions of the enumeration algorithm used for fixed polyominoes.

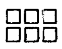
The general strategy used to enumerate symmetric polyominoes with a given simple symmetry is to identify each point in the field with its image under the symmetry's transformation. Points in the axis of symmetry, or at the centre, are already their own image. Whenever a cell is placed at a point, one is placed at the point's image. Similarly, whenever a point becomes a new neighbour, its image does too. But, of course, the new neighbour and its image are the *same* new neighbour, and they become one entry in the untried set. In general, each point and its image are represented by a unique representative (usually the one in the upper half of the field). When calculating the size of a polyomino, however, each cell and its distinct image(s) are counted separately. Thus the generated polyominoes will be invariant under the transformation (that is, symmetric) *by construction*.

The first step in the analysis is to break the primed functions down further, based on whether an axis of symmetry or centre is at a point, or is between points. For example, consider the HVADR2 polyomino and the two V polyominoes



The centre of the HVADR2 polyomino is a point, while the axes of symmetry of the two V polyominoes run through and between (columns of) points, respectively. If a symmetric polyomino has a centre at a point, or an axis of symmetry running through points, we add an "I" to the symmetry name. Similarly, if the centre is between points, or the axis runs between rows or columns of points, we add an "X" to the name. Thus, the example polyominoes have, respectively, HVADR2I, VI, and VX symmetry.

HVR and R symmetries have a centre that can be independently "I" or "X" in the vertical and horizontal dimensions. For example,



has HVRXI symmetry since the centre is between rows but inside a column. Note that the X symmetries are mutually exclusive of the I symmetries in each dimension.

The following list shows how "X" and "I" can be combined with fixed symmetries.

N has neither axis nor centre,
H = HX + HI,
V = VX + VI,
R = RXX + RXI + RIX + RII,
A and D have an axis which runs both between and through cells,
HVR = HVRXX + HVRXI + HVRIX + HVRII,
R2 = R2X + R2I,
ADR = ADRX + ADRI,
HVADR2 = HVADR2X + HVADR2I.

Now let us look at each symmetry function individually.

HX′($p$) is zero if $p$ is odd since every cell in an HX′ polyomino implies its image's existence, and a cell must be distinct from its image. When $p$ is even, HX′($p$) equals N′($\frac{1}{2}p$): and N′ polyomino can be turned into an HX′ polyomino of twice the size by reflecting the original at its bottom edge, and the reverse operation of slicing an HX′ polyomino along its axis yields an N′ polyomino.

To enumerate HI′ polyominoes, we use the general strategy of identification outlined above. In our canonical form, the $x$-axis is the axis of symmetry, and the left-most cell on the axis is forced to be at the origin. For each pair of identified points, the one above the axis is taken as the representative. There is no interaction between the upper and lower half of the field so that only representative points are used. We forbid representative cells to be placed below the axis, or to the left of the origin on the axis. It turns out that the HI′ enumerator is identical to the fixed enumerator except in calculating the size of polyominoes produced: each cell off the $x$-axis represents two; each on the axis represents only one.

VX′($p$) is equal to HX′($p$) and VI′($p$) is equal to HI′($p$), of course.

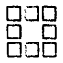
A′($p$) is computed by a modified version of the HI′($p$) enumerator. The change is a kind of counterclockwise rotation by $\frac{1}{4}\pi$ of the field. Thus, the diagonal takes the place of the $x$-axis. The initial cell is the same: the origin. The canonical form of an A′ polyomino has the lowest cell on the diagonal of symmetry at the origin.

D′($p$) equals A′($p$).

In a rotationally symmetric polyomino, only a cell at the centre can be its own image. Thus rotationally symmetric polyominoes have an odd size if and only if they have a centre cell. Let us consider the enumeration of odd-sized and even-sized rotationally symmetric polyominoes separately.

All odd-sized rotationally symmetric polyominoes have at least RII symmetry since the centre must be a cell, and therefore not between points. The canonical form of RII′ polyominoes has the centre cell at the origin. Unlike previous enumerators, growth in the upper and lower halves of the field interact, so we must actually place image cells, and mark unfree image new-neighbours. No cells need be forbidden to force the canonical form.

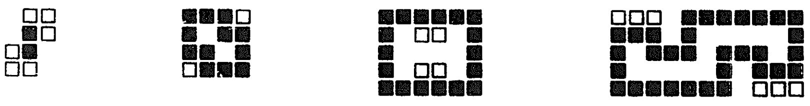
The enumeration of even-sized rotationally symmetric polyominoes is the most complex. In every other case, polyominoes are grown, starting from a polyomino with one cell (or one cell plus its image) having the desired property. Unfortunately, not all even-sized rotationally symmetric polyominoes contain an appropriate subpolyomino. For example, removing any cell plus its image from



causes it to be disconnected, and thus it cannot be grown from a smaller polyomino.

A simple way to enumerate even sized rotationally symmetric polyominoes would be to grow symmetric collections of cells, and count each one only if it is connected. Unfortunately this method is quite slow because most configurations generated would not be connected. We use a faster and more complicated method.

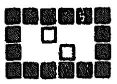
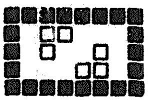
Every rotationally symmetric polyomino contains a collection of symmetric subpolyominoes sharing its centre. This collection is never empty since it at least contains the original polyomino. Some of these subpolyominoes are minimal in the sense that removing any cell and its image would disconnect the polyomino. Of these minimal subpolyominoes, there is exactly one that surrounds the minimum number of points. This subpolyomino is called the *ring*. In odd-sized rotationally symmetric polyominoes, the ring is the centre cell, but in even-sized polyominoes the ring may be quite large. Here are a few polyominoes with their ring cells denoted by ■



As these examples demonstrate, rings surround a (possibly empty) part of the field, hence their name. It is also notable that each cell in a ring touches exactly two others in the ring, except in degenerate rings of one or two cells. If this were not the case, some cell and its image could be removed without cutting the ring.

Our enumeration method generates each possible ring, and then from these it grows each possible rotationally symmetric polyomino. Once the ring is generated, connectivity need never be a problem since each new cell may be placed adjacent to the rest of the polyomino. What is important, however, is that the ring from which a polyomino is grown must be *the* ring of that polyomino, so that each polyomino is generated only once.

To ensure this last property, growth inside the ring is restricted to ensure that it does not make any part of the ring removable. All growth inside the ring must be connected to the ring. Any subpolyomino inside the ring that touches the ring must touch it at one edge, or at most two edges that are co-linear and adjacent, in order not to "short-circuit" the ring. Here are two examples of filled rings.

The methods used to grow rings and then their offspring are fairly straightforward.

All remaining symmetries are composite. Several can be computed directly from these relations:

$$\begin{aligned}
&\mathrm{HVRXX}'(2p) = \mathrm{HX}'(p),\\
&\mathrm{HVRXI}'(2p) = \mathrm{HI}'(p),\\
&\mathrm{HVADR2X}'(4p) = \mathrm{A}'(p).
\end{aligned}$$

Unmentioned values of these functions are zero. Other composite symmetries are enumerated by checking polyominoes generated in simple symmetry enumerators for additional symmetry.

## 7. Results

The fixed enumerator has been coded in fewer than fifty statements of the ALGOL W programming language. It has also been coded in the assembly language of the PDP-11 computer. This version has been run for polyominoes up to size 24, taking ten months of CPU time on a PDP-11/70.

The symmetric enumerators have been coded in ALGOL W, and run up to size 25. The total CPU time was less than five minutes on an IBM 370 model 165. This confirmed our expectation that fixed($p$) would be the dominant computation.

Table 2 lists the values of free($p$) and fixed($p$) for $p$ up to 24. Table 3 lists the number of polyominoes of each free symmetry type up to size 25 (except for none(25), which requires fixed(25)).

The values agree with those Lunnon computed except for size 17. In several runs the program has computed the value of fixed(17) to be 400795844 versus Lunnon's value of 400795860. We seem to have two fewer asymmetric free polyominoes since all symmetric counts agree. If the error is ours, it must be in our calculation of fixed(17), not in our symmetric enumerators. Because of the way our program works, it is virtually impossible to calculate fixed(17) incorrectly and fixed(18) correctly. Lunnon described how an undetected machine malfunction had caused other results of his to be wrong. He noticed the mistakes due to

Table 2

| | free | fixed | CPU hrs |
|---|---|---|---|
| 1 | 1 | 1 | — |
| 2 | 1 | 2 | — |
| 3 | 2 | 6 | — |
| 4 | 5 | 19 | — |
| 5 | 12 | 63 | — |
| 6 | 35 | 216 | — |
| 7 | 108 | 760 | — |
| 8 | 369 | 2725 | — |
| 9 | 1285 | 9910 | — |
| 10 | 4655 | 36446 | — |
| 11 | 17073 | 135268 | — |
| 12 | 63600 | 505861 | — |
| 13 | 238591 | 1903890 | — |
| 14 | 901971 | 7204874 | — |
| 15 | 3426576 | 27394666 | — |
| 16 | 13079255 | 104592937 | 0.148 |
| 17 | 50107909 | 400795844 | 0.560 |
| 18 | 192622052 | 1540820542 | 2.138 |
| 19 | 742624232 | 5940738676 | 8.196 |
| 20 | 2870671950 | 22964779660 | 31.409 |
| 21 | 11123060678 | 88983512783 | 120.957 |
| 22 | 43191857688 | 345532572678 | 467.053 |
| 23 | 168047007728 | 1344372335524 | 1807.263 |
| 24 | 654999700403 | 5239988770268 | 6959.665 |

large discrepancies with predicted values. In this case, the discrepancies are miniscule. For these reasons, we believe our result to be correct.

In one run on a PDP-11/70 the program computed fixed($p$) for $p$ up to and including 18 in two hours. This compares well with Lunnon's 175 hours, although his machine (an ATLAS I) was somewhat slower. Probably the main reason for our program being faster is that it need not check for symmetry or canonicity of generated polyominoes.

It seems unlikely that any technique that actually generates every polyomino could get much farther than ours. The fixed enumerator generates polyominoes at better than one every five microseconds. Future work on polyomino enumeration may be aimed at calculating fixed($p$), since we have shown how to compute free($p$) relatively cheaply given fixed($p$).

## Acknowledgments

R.C. Read introduced me to the problem, and suggested I publish my results. Many people have helped me find spare computer time. R.C. Read, D.A. Klarner, T.B. Rushworth, and especially W.D. Elliot deserve thanks for their constructive criticisms of this paper.

Table 3

| | all | axis 2 | rotate 2 | diag 2 | axis | rotate | diag | none |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | | | | | | | |
| 2 | | 1 | | | | | | |
| 3 | | 1 | | | | | 1 | |
| 4 | 1 | 1 | | | 1 | 1 | | 1 |
| 5 | 1 | 1 | | | 2 | 1 | 2 | 5 |
| 6 | | 2 | | | 6 | 5 | 2 | 20 |
| 7 | | 3 | | 1 | 9 | 4 | 7 | 84 |
| 8 | 1 | 4 | 1 | 1 | 23 | 18 | 5 | 316 |
| 9 | 2 | 4 | | | 38 | 19 | 26 | 1196 |
| 10 | | 8 | | 1 | 90 | 73 | 22 | 4461 |
| 11 | | 10 | | 2 | 147 | 73 | 91 | 16750 |
| 12 | 3 | 15 | 3 | 3 | 341 | 278 | 79 | 62878 |
| 13 | 2 | 17 | 2 | 3 | 564 | 283 | 326 | 237394 |
| 14 | | 30 | | 5 | 1294 | 1076 | 301 | 899265 |
| 15 | | 35 | | 6 | 2148 | 1090 | 1186 | 342211 |
| 16 | 5 | 60 | 12 | 14 | 4896 | 4125 | 1117 | 13069026 |
| 17 | 4 | 64 | 7 | 9 | 8195 | 4183 | 4352 | 50091095 |
| 18 | | 117 | | 20 | 18612 | 15939 | 4212 | 192583152 |
| 19 | | 128 | | 20 | 31349 | 16105 | 16119 | 742560511 |
| 20 | 12 | 236 | 44 | 56 | 70983 | 61628 | 15849 | 2870523142 |
| 21 | 7 | 241 | 25 | 32 | 120357 | 62170 | 60174 | 11122817672 |
| 22 | | 459 | | 80 | 271921 | 239388 | 60089 | 43191285751 |
| 23 | [illegible] | 476 | | 64 | 463712 | 240907 | 226146 | 168046076423 |
| 24 | 20 | 937 | 165 | 224 | 1045559 | 932230 | 228426 | 654997492842 |
| 25 | 11 | 912 | 90 | 114 | 1792582 | 936447 | 854803 | ? |

## References

[1] S.W. Golomb, Polyominoes (Charles Scribner's, New York, 1965).

[2] R.C. Read, Contributions to the cell growth problem, Canad. J. Math. 14 (1962) 1–20.

[3] D.A. Klarner, Cell growth problems, Canad. J. Math. 19 (1967) 851–863.

[4] D.A. Klarner and R. Riverst, A procedure for improving the upper bound for the number of $n$-ominoes, Canad. J. Math. 25 (1973) 585–602.

[5] W.F. Lunnon, Counting polyominoes, in: A.O.L. Atkin, B.J. Birch, eds., Computers in Number Theory (Academic Press, London, 1971) 347–372.