A method for cutting squares into distinct squares

Ian Gambini

Laboratoire d'Informatique de Marseille - Université de la Méditerranée - Faculté des Sciences de Luminy - 163, avenue de Luminy, Case 901 - 13288 Marseille Cedex 9 - FRANCE

Abstract

We are interested in dissecting squares into distinct squares. We impose the condition that the squares have integer sizes. This restriction does not reduce the number of solutions since it is always possible to scale a non-integer solution to obtain an integer one. This leads, when the square size n is fixed, to a finite enumeration.

We propose an algorithm which enumerates a subset of solutions when n is fixed. In spite of the incompleteness of this algorithm, we find many solutions and, in particular, for each integer p between 21 and 63, we have a solution using p squares.

A second algorithm, this time complete, but computationally prohibitive, allows us to find an unexpected result: the smallest decomposition using integer squares has a size of 110.

This paper is devoted to the description of the algorithms and to the presentation of an interesting subset of solutions.

Key words: squaring squares; perfect squares; squaring

1 Introduction

A dissection is called *perfect* when all the squares have distinct sizes and the *order* is the number of elements (squares) used in the dissection. Historically, the first perfect dissection was described by R. Sprague in 1939 [11]. Since then, many solutions have been found, the problem being to find the one with the smallest order. C.J. Bouwkamp, A.J.W. Duijvestijn and P. Medema [6] demonstrate that there is no possible dissection if the order is less than 21. In 1948, a perfect square of order 24 was found by T.H. Willcocks [15] (figure 2), followed in 1964 by J.S. Wilson [13] with a perfect square of order 25. In 1967, five perfect squares of order 25 were published by J.S. Wilson in his thesis [16].

Then, three other perfect squares of order 25 were obtained by P.J. Federico in 1978 [10]. Finally, A.J.W. Duijvestijn [7] found a solution of order 21 (figure 1) the March 22, 1978 with the aid of a DEC-10 computer. This solution is the single one of minimal order (except for symmetries).

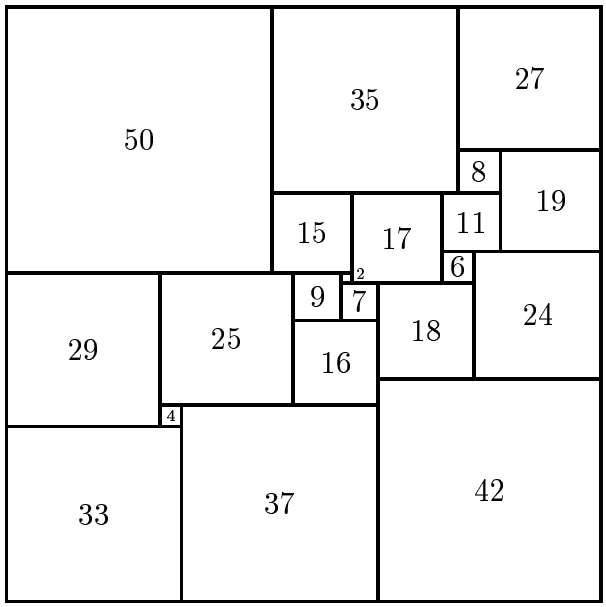


Fig. 1. The perfect square of order 21 (112×112)

The same kind of result has been proved for perfect *compound* squares [9], in which a strict subset of elements is arranged in the form of a rectangle. Figure 2 shows the only perfect *compound* solution of minimal order (order 24), except for symmetries.

The method we describe is an heuristic algorithm which proceeds by enumeration of integer values. Those values are the sizes of the squares in the dissection, so that we only obtain solutions with integer square sizes. This is not really problematic because it is possible to show that we can always scale a solution to obtain integer sizes as follows: C. Berge [2] shows that each solution of this problem can be expressed as the only solution in the reals of a system of linear equations with integer coefficients, so that the solutions are all rationals. Therefore, by multiplying those sizes by a coefficient, we obtain integer solutions.

2 Our enumerative algorithm

The main idea of the method is motivated by another question: how can we find the perfect solution of order 21? Reconsider the figure 1 and suppose that we know the bottom of the solution.

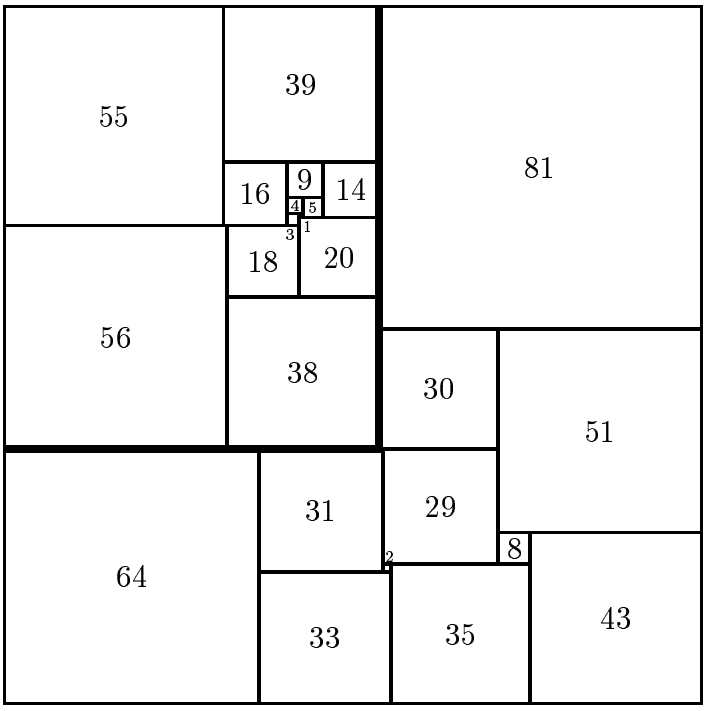


Fig. 2. Perfect compound square of order 24 (175×175)

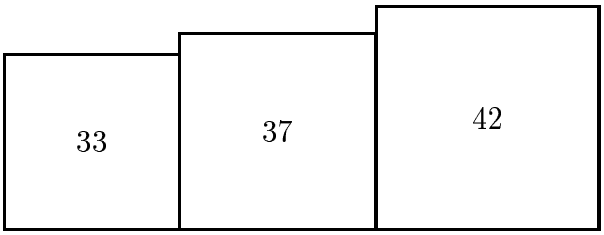


Fig. 3. Initial Configuration.

Starting with this configuration (figure 3), it is possible to find the square of size 4 extending the top of the square of size 37 and 33, and then the one of size 29 by extending the left side of the square of size 4 and 33. This way, we finally find the solution of the 21 squares. The question is then to define the rules which allow us to know the size of each new square.

The idea consists of constructing the big square from the bottom to the top, piling up squares. From an algorithmic point of view, we only need to know the top line of the construction; this means that at each stage we only need to keep in memory a list of horizontal lines which we call *plates*.

Let h_i and l_i be the height and the width of the plate i, taking the lower left corner as origin. In such a case, we can define two simple rules for building squares:

- The *horizontal extension* (figure 4), which consists in adding a square of size $h_{i+1} - h_i$ if $h_{i+1} > h_i$ and of size $h_i - h_{i+1}$ if $h_i > h_{i+1}$.

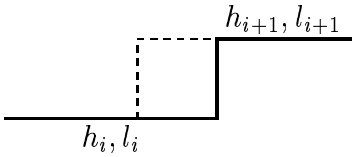


Fig. 4. Horizontal extension

- The *vertical extension* (figure 5), which consists in placing a new square on the full length of the plate. This way, we add a square of size l_i.

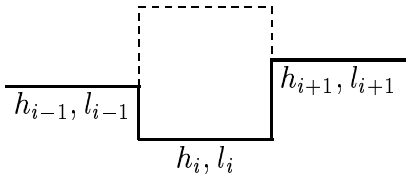


Fig. 5. Vertical extension

This way, applying these two rules to the initial configuration of the figure 3, we rapidly obtain the solution of order 21. The advantage of this method lies in its simplicity. The number of choices at each stage is small, and starting with an initial configuration the solution is completed rapidly.

The main disadvantage is the initialization with its prohibitive enumeration. Effectively, we have to fix the number of squares at the bottom and we also have to fix their sizes, which is very expensive in terms of enumeration. This method is also limited because we can find many solutions but not all of them.

For example, we cannot find the solution of order 24 (figure 2).

3 Details of the algorithm

The implementation of this method is, however, delicate because we have to take care not to process the same configuration more than once. A stage of the construction is made of two pieces of information:

(1) The knowledge of the square sizes already used, to avoid using the same size again, since we are looking for *perfect* dissections.
(2) The knowledge of the top of the construction is sufficient for using our construction rules. A possible codification could be to store a list of a couple (height, width) corresponding to each plate.

We have to deal with fixing an order for the placement of the squares because doing the search by placing first a square a and then a square b gives the same results as placing the square b and then the square a. To avoid finding

the same solutions by processing the same configuration more than once, we have to choose a method for placing the squares which avoids constructing the same configuration more than once. This rule was not easy to find because we have to take care not to forget some complicated particular cases.

What are the main stages of the algorithm? First, we have to enumerate the sizes for the square to be dissected. Next, to obtain an initial configuration, we have to enumerate the squares sizes at the bottom. The number of squares being enumerated at the bottom is a parameter of our algorithm.

There are some modifications to improve this method. First, to avoid vertical symmetry, we impose the condition that the bottom left square be smaller than the bottom right one. This way we obtain an initial configuration like the one in figure 6. We can also notice in this figure the adjunction of two plates corresponding to the top of the square to be dissected. Taking this configuration rather than the first one allows to find more solutions by extending those plates.

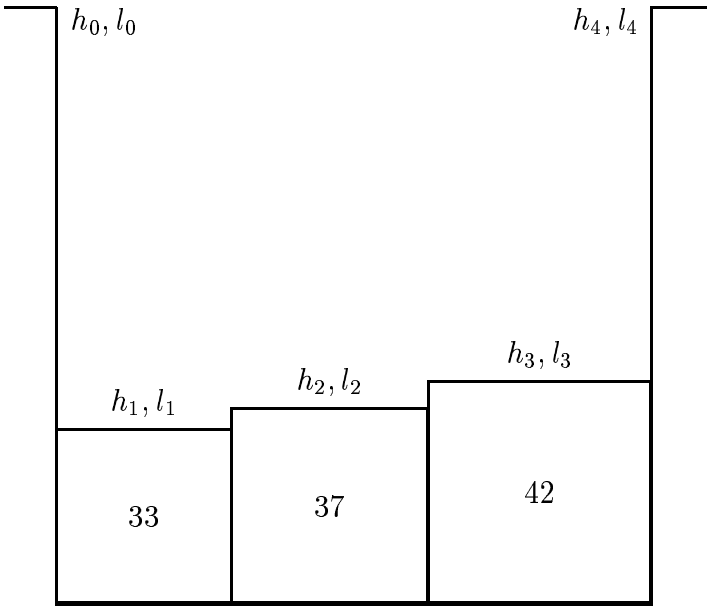


Fig. 6. Initial configuration of the algorithm.

With regard to the method for placing new squares, we proceed from left to right. Each time we place a new square on a plate i, we have to look to the modifications of this placement on adjoining plates; the next square will be placed first on the modified plate at the extreme left. This way, we avoid constructing the same configuration, so that time is saved and each solution is obtained only once.

An important property of this algorithm is that it cannot increase the number of plates during construction. This is true because our two construction rules (the horizontal extension and the vertical extension) do not create new plates,

they just modify the height (vertical extension) or the width (horizontal extension) of existing ones. This number of plates can only decrease and result in one plate when a solution of perfect square is found.

However, this algorithm can also find decompositions of perfect rectangles. To find them, we have just to look at the plates. When all the initial plates have the same height, making a single plate, we know that we have found a dissection of a perfect rectangle.

As a matter of fact, the two rules for the construction allow us to create new squares whose sizes are either sums or differences of sizes already used. So, if the squares used at the initialization stage have a greatest common divisor different from 1, each new square created by our rules will be also divisible by it. This will lead to a solution already found but with sizes multiplied by a constant. To avoid this, if the sizes of the initial squares have a greatest common divisor different from 1 we will not continue further.

This test does not allow us to eliminate enough configurations and does not save much time, but this technique allows us to find only minimal integer solutions.

Another remark can be made about this initial configuration. There are configurations which cannot give any solution. For example, the one in figure 7 cannot lead to a solution because our algorithm will never succeed in filling the space on the top of the square of size 15.

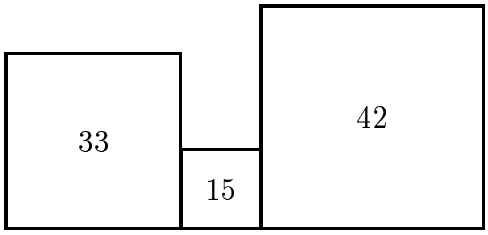


Fig. 7. Example of initial configuration without solution.

To generalize, for each plate i, if we take (h_{i-1}, l_{i-1}), (h_i, l_i), (h_{i+1}, l_{i+1}) as three consecutive plates resulted from initialization, then the configuration will have no solution if we have $l_i < h_{i-1} - h_i$ and $l_i < h_{i+1} - h_i$. This remark allows for an acceleration of our algorithm.

4 Pseudo-code

We give a pseudo-code of the algorithm described before. There are two main parts. The first one (procedure *solve*) is very simple and its aim is to generate

all the possible initial configurations described in figure 6 and for each configuration, we launch the resolution. This step requires two parameters which are the size of the square to be dissected (*square_size*) and the number of squares used at the bottom to create the initial configuration (*nb_base_squares*).

Procedure $solve(nb\_base\_squares, square\_size)$
  $h_0 = h_{nb\_base\_squares+1} = square\_size$
  $l_0 = l_{nb\_base\_squares+1} = 1$
  ⟨initialize enumeration⟩
  for each ⟨enumeration⟩ **do**
    call $horizontal\_extension(p_0, p_1)$

This first step generates the starting top line of the construction stored in two vectors called h_i and l_i representing the height and the width of the plate p_i ($i \in \{0 \dots nb\_base\_squares + 1\}$). Now, following the previous explanation of the algorithm and paying attention to some particular cases, we can construct the following procedures (*horizontal_extension* and *vertical_extension*) :

Procedure $horizontal\_extension(p_i, p_{i+1})$
  if $h_{i+1} > h_i$ **then**
    if $h_{i+1} - h_i < l_i \wedge size\_not\_used(h_{i+1} - h_i)$ **then**
      ⟨place the square of size $h_{i+1} - h_i$⟩
      if $h_{i-1} - h_i = l_i$ **then**
        if $\neg last\_plate(p_{i+1})$
          then CALL $vertical\_extension(p_{i+1})$
          else CALL $vertical\_extension(p_i)$
      ⟨remove the square⟩
  else if $h_i - h_{i+1} < l_{i+1} \wedge size\_not\_used(h_i - h_{i+1})$ **then**
    ⟨place the square of size $h_i - h_{i+1}$⟩
    if $\neg first\_plate(p_i)$
      then if $h_{i-1} \geq l_i + h_{i+1}$
        then CALL $horizontal\_extension(p_{i-1}, p_i)$
        else CALL $vertical\_extension(p_i)$
      else CALL $vertical\_extension(p_{i+1})$
    ⟨remove the square⟩
  if $\neg last\_plate(p_{i+1})$ **then** CALL $vertical\_extension(p_{i+1})$

These two procedures are the core of the algorithm and all the processing time is spent here. They correspond to the two main rules described before: the horizontal extension and the vertical extension. In this pseudo-code, we have used an array to represent the plates configuration but in a real implementation, it will be more efficient to use a linked list stored in an array.

Procedure $vertical\_extension(p_i)$
  if $size\_not\_used(l_i) \wedge h_i + l_i \leq square\_size$ **then**
    $\langle$save configuration$\rangle$
    $h' = h_{i-1}$
    $l' = l_{i-1}$
    $\langle$place the square of size $l_i\rangle$
    if $h_i = h_{i+1}$ **then** $\langle$merge plate p_{i+1} to $p_i\rangle$
    if $h_{i-1} = h_i$ **then** $\langle$merge plate p_{i-1} to $p_i\rangle$
    $\langle$test if solution found$\rangle$
    if $first\_plate(p_i)$
      then if $\neg last\_plate(p_i)$ **then** CALL $horizontal\_extension(p_i, p_{i+1})$
      else if $h_i - l_i - h_{i-1} = l_{i-1}$
        then CALL $vertical\_extension(p_{i-1})$
        else if $h' = h_i \wedge h_{i-1} - h_i \leq l'$
          then if $\neg last\_plate(p_i)$ **then** CALL $vertical\_extension(p_i)$
          else CALL $horizontal\_extension(p_{i-1}, p_i)$
    $\langle$restore configuration$\rangle$
  if $h_{i-1} - h_i \leq l_i \vee h_{i+1} - h_i \leq l_i$ **then** CALL $horizontal\_extension(p_i, p_{i+1})$

5 General results

An important parameter of our algorithm is the number of squares at the bottom of the construction. This number is also the number of plates at the beginning. Our choice was first to use three plates to find the solution of order 21. We can also start the algorithm with a different number of plates, but using three plates we rapidly obtained many solutions.

The results obtained using three plates and enumerating the size of the square to be dissected from 6 to 1200 are as follows:

- 2,718 perfect squared squares (not taking symmetries into account);
- 5,961 perfect squared rectangles (not taking symmetries into account);
- the only *compound* perfect square found is the one of size 196 of order 52 (cf. set of solutions);
- the first perfect square found is the solution of order 21 of the figure 1 after 2.5 seconds;
- 115,070,342 initial configuration processed out of 138,143,297 enumerated, on which 544,802,604,910 squares have been placed;
- the calculation took 143 hours of processing time using a Pentium Pro 200 under Linux.

Figure 8 shows us the distribution of the orders in the solutions found. We can observe a curve looking like a "bell" which extends to a higher order when we continue searching. This allows us to think that the bigger the size of the

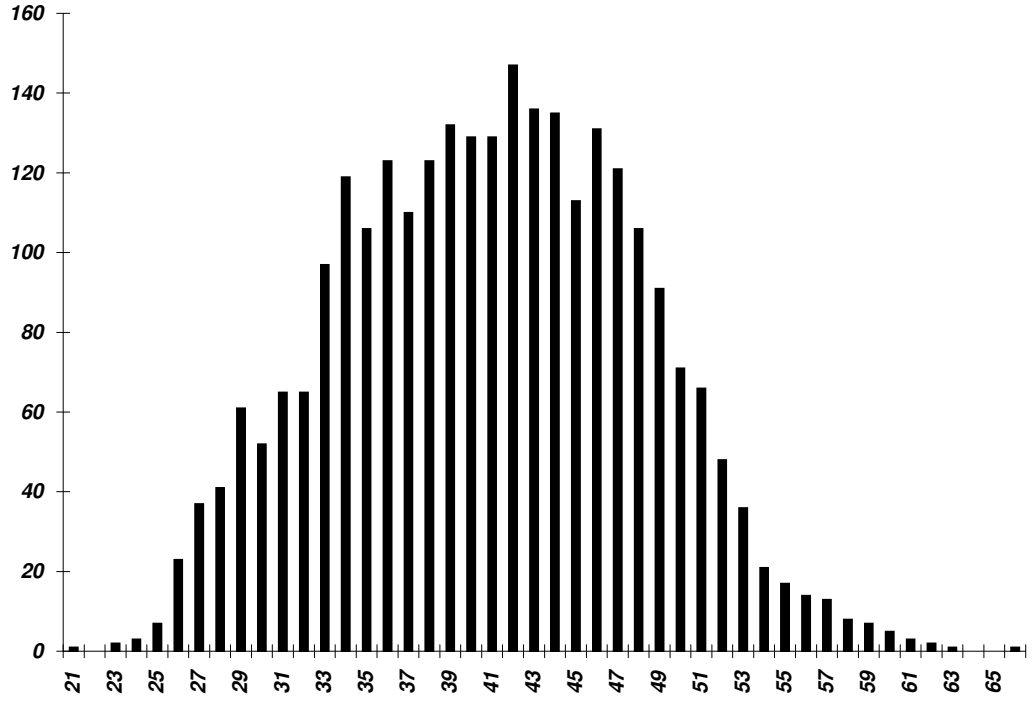


Fig. 8. Order distribution of squares found (using 3 plates).

square to dissect, the better the chances of finding higher order solutions. A strange result was that our algorithm did not find any solution of order 22. This is true only if we start using 3 plates; but using 4 or 5 plates we find fewer solutions but we find two squares of order 22 (figure 9).

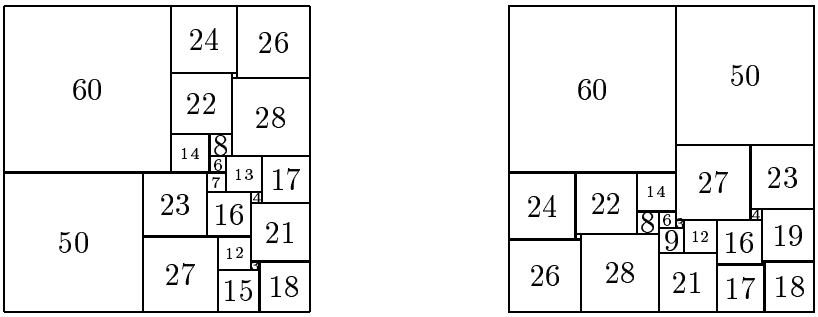


Fig. 9. Two perfect squares 110×110 of order 22

Having found squares of order 22, we have therefore found solutions from order 21 to 63. If we take the square of order 21 and if we dissect this smallest square into 21 distinct squares, we obtain a new dissection of order 41 but this time the solution is *compound*. So, using dissections from order 21 to 40, we can construct a new solution of any order greater or equal to 21.

If we examine the existence of minimal integer solutions, the squares of figure 9 are interesting: they are the smallest ones (110×110) found by our algorithm. The question is then to know if 110 is the smallest size for integer solutions,

and if the two squares found are the only ones.

To test that, we have first shown the following property, by an enumeration of all possible cases, which allows us to have a reasonable processing time:

Property 1 *Squares of sizes* 1, 2, 3 *or* 4 *cannot appear on the sides of a dissection using integer sizes.*

The proof of this property is pretty simple to do manually but it takes a little bit long to explain it here. The idea of the method is the following: if the square of size 1 is placed at the bottom of a construction, since the square at the left side and the square at the right side are bigger, the only way to continue the construction is to place a square at the top of the square of size 1. The only way to do it is to place a new square of size 1 which is forbidden because all squares must have different sizes. So you cannot have a square of size 1 on the sides of an integer dissection. Similarly using squares of sizes 2, 3 and 4 one can complete the proof of property 1.

6 The second algorithm

To find the smallest square which can be dissected using squares of integer sizes, we have modified our first algorithm to make a new one.

This new algorithm is simpler but the space search is prohibitive. Instead of using our two preceding rules, we now have a unique rule to apply at each stage of the algorithm: we consider all the possible configurations on the top of the smallest delimited plate.

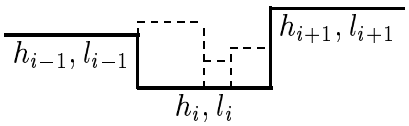


Fig. 10. Delimited plate

We take the smallest plate to reduce the search space and we take only delimited plates to enumerate all possible configurations leading to a possible solution. If the plate is not delimited, we don't know the definitive width of this plate because a horizontal extension is possible. This way, we are sure to find all possible solutions because at each stage we enumerate all the possible following configurations leading to a solution.

The initialization of this algorithm is simpler than the previous one. Here, we start with a single plate at the bottom having the width of the square we want do dissect.

In fact, instead of dissecting a plate, the basic rule can be simpler. At each stage, we place a new square at the left of the smallest plate delimited. This method is quite the same than before but is simpler to implement.

6.1 Pseudo-code

As said before, the starting procedure ($solve2$) is really simple. For technical reasons, we have to place two virtual plates to delimit the construction, otherwise the first plate will not be delimited as required for its decomposition. This way we are sure to always find a smallest delimited plate.

Procedure $solve2(square\_size)$
  $h_0 = h_2 = square\_size + 1$
  $h_1 = 0$
  $l_0 = l_2 = 1$
  $l_1 = square\_size$
  CALL $decompose(p_1)$

The next step is to find the smallest plate delimited and start its decomposition (procedure $next\_plate$). If the plate width is equal to the square size then we know that we have found a solution.

Procedure $next\_plate()$
  $l_{min} = square\_size + 1$
  $i = 1$
  while $\neg last\_plate(p_i)$ **do**
    if $h_{i-1} > h_i \wedge h_i < h_{i+1} \wedge l_i < l_{min}$ **then**
      $p_{min} = p_i$
      $l_{min} = l_i$
    $i = i + 1$
  if $l_{min} = square\_size$
    then
      if $h_{min} = square\_size$
        then ⟨square found⟩
        else
          ⟨rectangle found⟩
          CALL $decompose(p_{min})$
    else CALL $decompose(p_{min})$

Finally, the most complex procedure is the decomposition of a plate (procedure $decompose$). In fact, the following procedure does not really decompose a plate. Its aim is just to place the first square of the plate decomposition.

There are different cases: first we use the whole plate width, then we try to extend the previous plate (i.e. horizontal extension) and finally we enumerate all possible square sizes.

In the first case, we have finish decomposing the plate so we have to find a new plate to decompose. In the last two cases, the plate decomposition is not finished so we continue the algorithm decomposing the remaining plate width.

Procedure *decompose*(p_i)
  if $size\_not\_used(l_i) \wedge h_i + l_i \leq square\_size$ **then**
    ⟨save configuration⟩
    ⟨place the square of size l_i⟩
    if $h_i = h_{i+1}$ **then** ⟨merge plate p_{i+1} to p_i⟩
    if $h_i = h_{i-1}$ **then** ⟨merge plate p_i to p_{i-1}⟩
    CALL *next_plate*()
    ⟨restore configuration⟩
  if $size\_not\_used(h_{i-1} - h_i) \wedge h_{i-1} - h_i < l_i$ **then**
    ⟨place the square of size $h_{i-1} - h_i$⟩
    CALL *decompose*(p_i)
    ⟨remove the square⟩
  for $size = 2$ **to** $MIN(l_i - 1, square\_size - h_i)$ **do**
    if $size\_not\_used(size) \wedge size \neq h_{i-1} - h_i$ **then**
      ⟨place the square of size *size* on p_i⟩
      CALL *decompose*(p_{i+1})
      ⟨remove the square⟩

The procedure *decompose* can be optimized because the second test is useful only for the first square of the plate decomposition.

6.2 Results

We know that we can find all the dissections having a given size. The only problem is the extreme complexity of this new algorithm. We have tested this program to find all squares (and rectangles) of size less or equal to 112×112. Here are the results:

- 4 perfect squared squares (not taking symmetries into account), 3 of size 110×110 and 1 of size 112×112;
- 118 perfect squared rectangles (not taking symmetries into account);
- 1,633,183,723,812 squares have been placed;
- the calculation took 360 hours of processing time using a Pentium Pro 200 under Linux.

This new algorithm gave us not only the two solutions of figure 9 but also a new square of size 110 using 23 squares (figure 11).

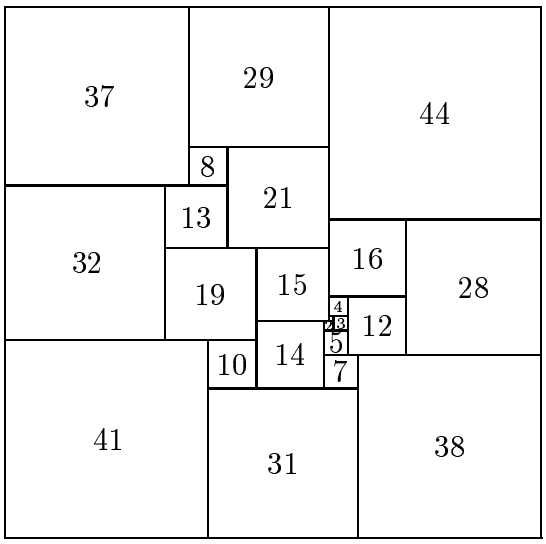


Fig. 11. Perfect square of size 110, using 23 squares

This square gives us the third and last square of size 110 (not found by our first algorithm).

We have thus shown three properties:

Property 2 *The smallest perfect square using integer sizes has a size of* 110.

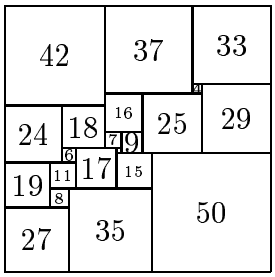
Property 3 *The set of perfect squares of size* 110 *using integer sizes include only two solutions of order* 22 *(figure 9) and one of order* 23 *(figure 11).*

Property 4 *The square of order 21 (figure 1) is the only one of size* 112×112.

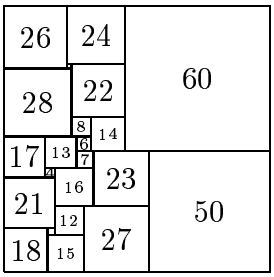
7 Interesting subset of solutions

For every order from 21 to 63, we give the first solution found by the algorithm, which corresponds to squares with minimal sizes. Among them, three are interesting:

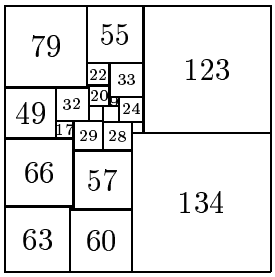
- the square 112×112 of order 21 is the one with the minimal order;
- the one 110×110 of order 22 is the first one of minimal size found;
- the one of size 976 of order 52, because at the present time, the algorithm has not found other *compound* perfect squares.



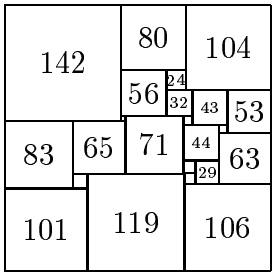
112 × 112, 21 Squares



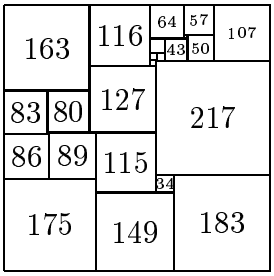
110 × 110, 22 Squares



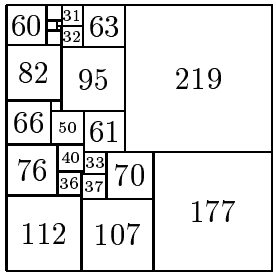
257 × 257, 23 Squares



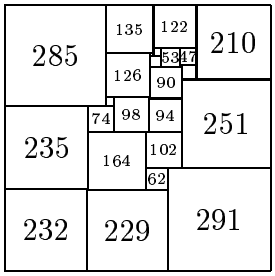
326 × 326, 24 Squares



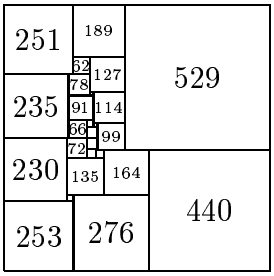
507 × 507, 25 Squares



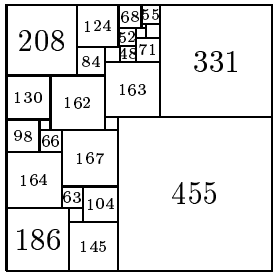
396 × 396, 26 Squares



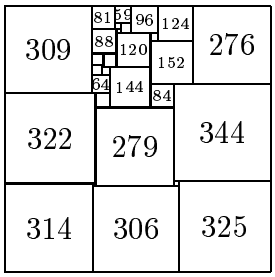
752 × 752, 27 Squares



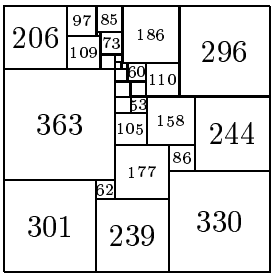
969 × 969, 28 Squares



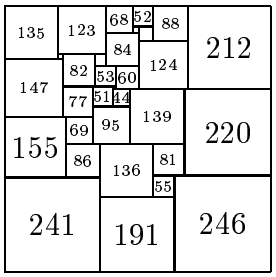
786 × 786, 29 Squares



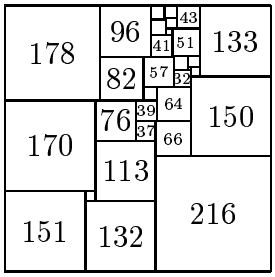
945 × 945, 30 Squares



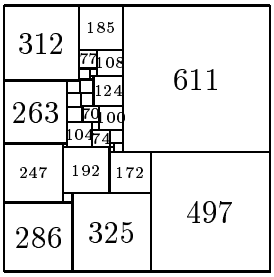
870 × 870, 31 Squares



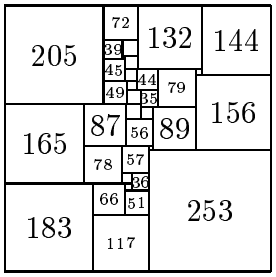
678 × 678, 32 Squares



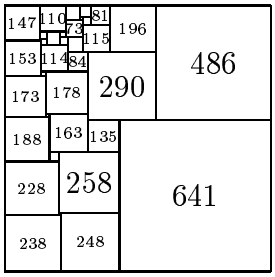
499 × 499, 33 Squares



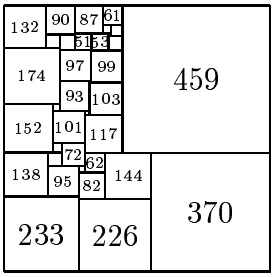
1108 × 1108, 34 Squares



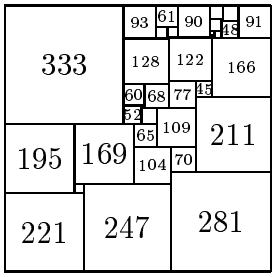
553 × 553, 35 Squares



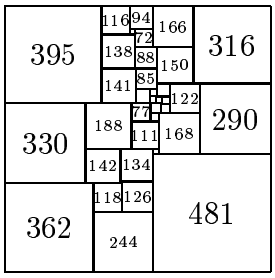
1127 × 1127, 36 Squares



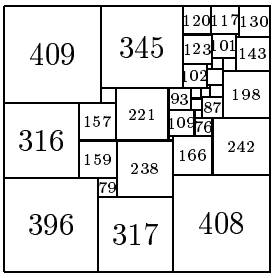
829 × 829, 37 Squares



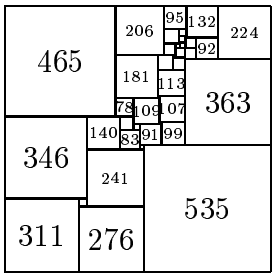
749 × 749, 38 Squares



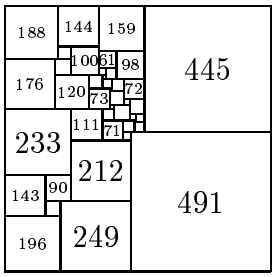
1087 × 1087, 39 Squares



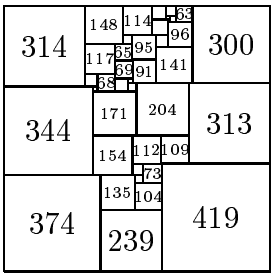
1121 × 1121, 40 Squares



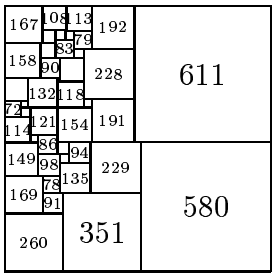
1122 × 1122, 41 Squares



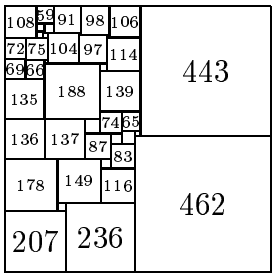
936 × 936, 42 Squares



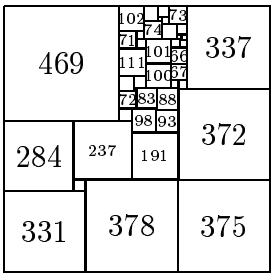
1032 × 1032, 43 Squares



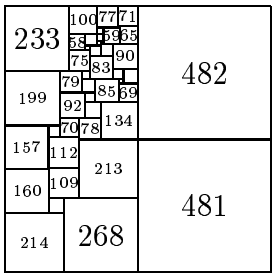
1191 × 1191, 44 Squares



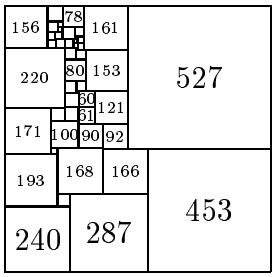
905 × 905, 45 Squares



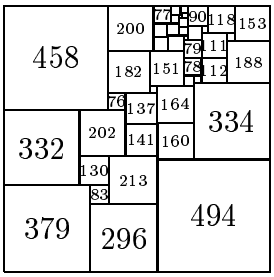
1084 × 1084, 46 Squares



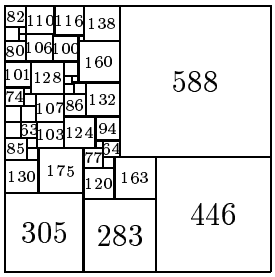
963 × 963, 47 Squares



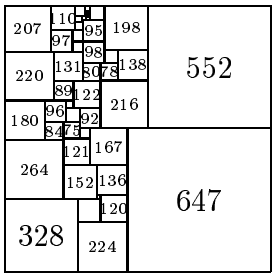
980 × 980, 48 Squares



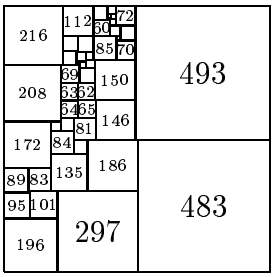
1169 × 1169, 49 Squares



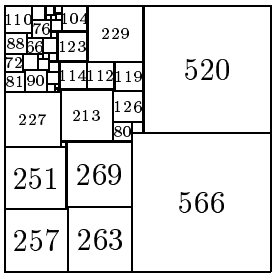
1034 × 1034, 50 Squares



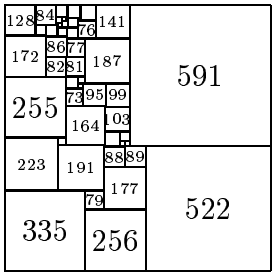
1199 × 1199, 51 Squares



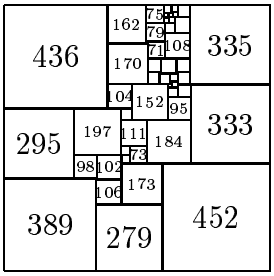
976 × 976, 52 Squares



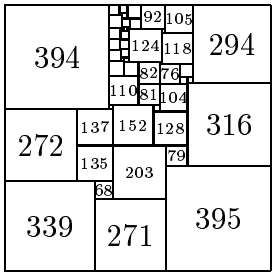
1086 × 1086, 53 Squares



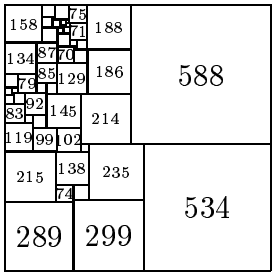
1113 × 1113, 54 Squares



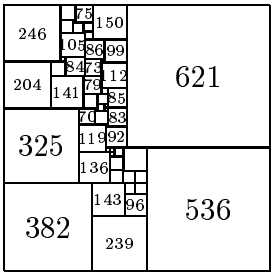
1120 × 1120, 55 Squares



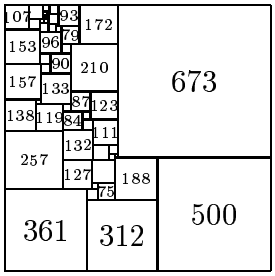
1005 × 1005, 56 Squares



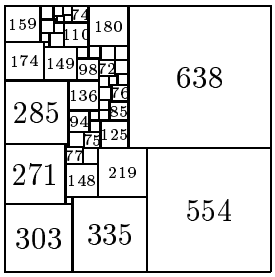
1122 × 1122, 57 Squares



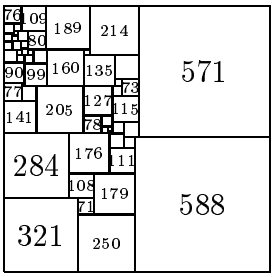
1157 × 1157, 58 Squares



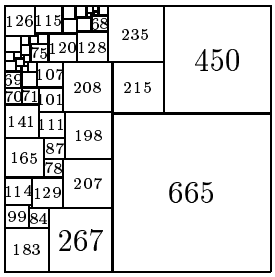
1173 × 1173, 59 Squares



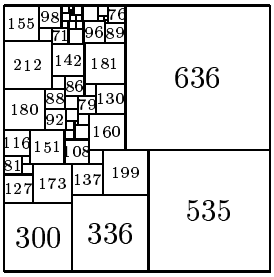
1192 × 1192, 60 Squares



1159 × 1159, 61 Squares



1115 × 1115, 62 Squares



1171 × 1171, 63 Squares

8 Conclusion

Until now, the vast majority of the results in this domain have been found using computer techniques, and mainly graph enumeration [12,14]. This approach was used to find the well known 21 squares solution [7] and also to find similar problems like Simple Perfect Square-Cylinders [1] and Simple Perfect 2×1 Squared Rectangles [3,8].

Another interesting method is to use constraint techniques to solve this problem [4]. This approach was also used to solve perfect square puzzles (where each square size is fixed) [5].

The benefit of the algorithm described here is that we can find squares and rectangles dissection rather rapidly and numerously. It seems that this method places squares astutely since we find optimal solutions for the order 21 and for the size 110×110. In addition, the solutions found are extremely rarely compound squares but it is easy to create compound dissections using non-compound ones.

References

[1] A. Augusteijn and A.J.W. Duijvestijn, Simple Perfect Square-Cylinders of Low Order, *Journal of Combinatorial Theory*, Series B 35, pages 333-337, 1983.

[2] C. Berge, Notice sur l'utilisation de la notion de potentiel pour les réseaux de transport. *Théorie des graphes et ses applications*, pages 240-249, Dunod, 1958.

[3] R.L. Brooks, A Procedure for Dissecting a Rectangle into Squares, and an Example for the Rectangle Whose Sides Are in the Ratio 2:1, *Journal of Combinatorial Theory 8*, pages 232-243, 1970.

[4] A. Colmerauer, An Introduction to Prolog III, *CACM*, pages 25-30, 28(4):412-418, 1990.

[5] Y. Colombani, Carrés Parfaits & Contraintes sur Intervalles d'Entiers, Mémoire de D.E.A., Faculté des Sciences de Luminy, Université de la Méditerranée, 1993.

[6] A.J.W. Duijvestijn, Electronic Computation of Squared Rectangles, Thesis, *Technological University, Eindhoven, The Netherlands*, 1962 ; *Philips Research Reports* **17**, pages 523-612, 1962.

[7] A.J.W. Duijvestijn, Simple Perfect Squared Square of Lowest Order, *Journal of Combinatorial Theory*, Series B 25, pages 240-243, 1978.

[8] A.J.W. Duijvestijn, A Lowest Order Simple Perfect 2×1 Squared Rectangle, *Journal of Combinatorial Theory*, Series B 26, pages 372-374, 1979.

[9] A.J.W. Duijvestijn, P.J. Federico and P. Leeuw, Compound Perfect Squares, *American Math. Monthly*, vol 89, pages 15-32, 1982.

[10] P.J. Federico, Squaring rectangles and squares: A historical review with annotated bibliography, in Graph Theory and Related Topics, *Academic Press*, pages 173-196, 1979.

[11] R. Sprague, Beispiel einer Zerlegung des Quadrats in lauter verschiedene Quadrate, *Math. Z.* **45**, pages 607-608, 1939.

[12] W.T. Tutte, A Census of Planar Maps, *Canad. J. Math. 15*, pages 249-271, 1963.

[13] W.T. Tutte, Squared rectangles, *Proceedings of the IBM Scientific Symposium on Combinatorial Problems*, pages 3-9, 16-18 March 1964.

[14] T.R.S. Walsh, Counting Non-isomorphic Three-Connected Planar Maps, *Journal of Combinatorial Theory*, Series B 32, pages 32-44, 1982.

[15] T.H. Willcocks, Problem 7795 and solution, *Fairy Chess Rev.* **7**, 1948.

[16] J.C. Wilson, A Method for Finding Simple Perfect Squared Squarings, Thesis, *University of Waterloo*, 1967.