A Look at Tiling Mutilated rectangles

Linda Hintzman
Oregon State University

Morgan Peavyhouse
Oregon State University

August 15, 1990

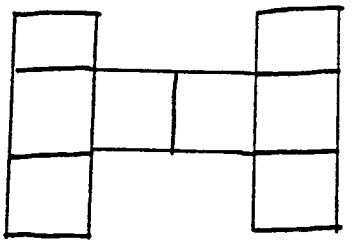
INTRODUCTION

The question that we examined this summer was how the removal of squares affected the number of domino tilings of a given region. The regions we concentrated on were rectangle of size m x n, where m or n had to be even. The dominoes used in our tiling were rectangles, 2 x 1 or 1 x 2 in size. The rules we followed in removing unit squares were as follows: 1. Color squares in region like a chess board; 2. Remove an equal number of red and black unit squares; 3. Remove only border squares; 4.Do not cause the region to become 2 or more disjoint regions. A border square is one that shares an edge with three or less other squares. Tiles that become border squares at any point during the removal of squares are then available for removal as long as their removal would not go against rule 4 above. The term mutilated is used to describe a region after the desired removals have occurred.

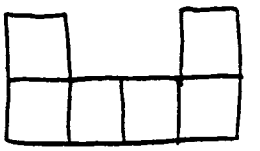
STARTING CONDITIONS

Looking at all possible regions is much too broad a question. As long as you start with a rectangle that has at least one even side you will be able to find one tiling of the original region. An even side assures an even area in which half the squares are red and half are black. Each domino covers one red and one black square. We need to start with a rectangle so that we are assured at least one tiling on the original area. There are regions that have equal numbers of red and black

squares that have no tilings but that can be mutilated so that a tiling of the new region does exist. For example:



No Tiling



One Tiling

GOAL

Our goal was to show that mutilating a region causes there to be fewer tilings of the new region than of the original region, or to find a counter example. We have not yet succeeded in this goal. Adapting methods for counting tilings of rectangles in Kasteleyn's paper we developed two methods of counting tilings of mutilated regions. The first involves creating a matrix and taking its determinant. The second creates permutation that stand for possible tilings and then counts the permutations that satisfy certain rules.

THE MATRIX WAY

In Kasteleyn's paper we discovered that the number of tilings of a rectangle is equal to the square root of the determinant of a matrix of 1's, 0's and -1's. The main drawback of this method is the size of the matrices involved. To look at a six by six square requires a 36 by 36 matrix. The dimensions of the matrix equal the area of the non mutilated region. The unit squares are numbered in ascending order starting at the bottom left corner, working across each row before going on to

the row above. For a 2 by 4 rectangle it would look like this:

5	6	7	8
1	2	3	4

Mathematica was used to create an off diagonal upper triangular matrix, A(i,j), and then subtract it's transpose from it to get the matrix we needed. Each entry in the matrix A(i,j) stands for the type of connection between two squares following these guidelines:

A(i,j) = 1 if it is possible to put a horizontal tile over squares i and j or if i and j are even and it is possible to cover them with a vertical tile.

A(i,j) = -1 if i and j are odd and it is possible to cover them with a vertical tile.

A(i,j) = 0 otherwise

For a two by four rectangle

$$A(i,j) = \begin{bmatrix} 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

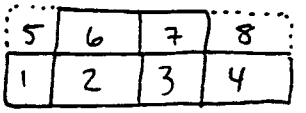
And D(i,j) = A(i,j) - Transpose(A(i,j))

$$D = \begin{bmatrix} 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 \end{bmatrix}$$

In this case the square root of the determinant of D equals five. Which is the number of ways of tiling either the two by four or

the four by two rectangle. The matrix for the four by two looks different but the square root of the determinant is the same.

Now if we wanted to look at a 2 by 4 rectangle minus the top corners,



the rules for the elements of A(i,j) are the same as above except that we connect the squares we have removed, make A(5,8) = 1, and make all other elements involving 5 or 8 equal to 0. So that A(1,5) = A(5,6) = A(7,8) = A(4,8) = 0. Just as before,

$$D = A - \text{Trans}(A) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The determinant of this new matrix equals one, there is only one way to tile the above shape.

When removing more than two border squares it does not matter which ones you choose to connect as long as they are not both red or both black, and that you have not previously chosen to connect either square to another square. If in the above example I had also removed three and four I could have connected (3 and 5) and (4 and 8) or (3 and 4) and (5 and 8). One thing we would like to determine is if it is possible to give the new connections the value 1 or if -1 is necessary in certain circumstances. In the few cases where odd squares were removed and connected the determinant was unchanged by the use of 1 or -1.

PERMUTATION WAY

Kasteleyn's paper also showed us how to use permutations to count tilings of non-mutilated rectangles. This approach runs into the same difficulty that the determinant did, that is size, because we have to find all the permutations of the numbers one through S, where S equals the area of the rectangle. We wrote a pascal program that finds all the permutations and then, following certain rules, erases permutations that do not correspond to a possible tiling. The rules for possible tilings are:

1. $n(1) < n(2); n(3) < n(4); n(5) < n(6); \ldots n(S-1) < n(S)$
2. $n(1) < n(3) < n(5) < \ldots < n(S-1)$
3. Either $n(i) = n(i+1)-1$ as long as $n(i)$ is not at the end of a row, or $n(i) = n(i+1) - r$, where r equals the number of columns and $n(i)$ is not in the top row.

We really don't have to find all the permutations. Due to rule 2, we only need permutations that begin with $n(1) = 1$. This still leaves us with a large number of permutations to compute and then check. Just looking at a four by four square gives us 15! permutations of the numbers 1 through 16 to deal with.

To use this program to count tilings of mutilated regions we would add rules that coincide to a particular mutilation. These rules would have to connect, one to one, the squares that were removed. (For a copy of the program see Appendix A)

NUMBERING SYSTEM

On a suggestion from Robert Burton (OSU) we would like to

examine the effects of an alternate numbering system. Instead of working across the rows from the bottom up, we work across on the diagonals. Start with 1 in the bottom left corner, then place 2 in the second from left bottom and work up and back along that diagonal. Once one diagonal is filled return to the bottom left most empty square. For a four by four square the new numbering looks like this

10	13	15	16
6	9	12	14
3	5	8	11
1	2	4	7

It has been suggested that this system might produce more symmetry when using trees to count tilings.

CONCLUSION

We plan to continue the work we have begun on this topic. Some more topics we would like to look into are:

1. Determine affect of mutilating rectangles on the number of tilings.
2. Find an upper bound for the number of tilings of similar types of mutilated regions.
3. Look at trees as a way of counting tilings.

REFERENCES

1) Grunbaum, B. and Shephard, G. C., Tilings and Patterns. Freeman and Company (1989).

2) Kasteleyn, P. W., The Statistics of Dimers on a Lattice. (1961).

3) Propp, J., Elkies, N., Kuperberg, G., Larsen, M., Alternating Sign Matrices and Domino Tilings. (1990).

```
uses printer;

const Sn = 8 ;
      Snf = 40320;     { ( Sn - 1 ) ! }
      Num_Col = 2;
      Num_Row = 4;

type  Parray = array [1..Snf,1..Sn] of integer;

var Per_array : Parray;
    Row,Column,J,Cycle,Length,Increment,Counter: integer;

function Factorial (Number:integer):integer;
 begin
  if Number>0 then Factorial:= Number*Factorial(Number-1)
  else Factorial:=1
 end;

function Match(Test,I,J:integer):boolean;
 begin
  if Test=Per_array[I,J] then
   Match:=true
  else Match:=false
 end;

procedure Look_write(var Test:integer; I,J:integer);
var changed : boolean;
begin
 changed := true;
 while changed do
 begin
  changed:=false;
  J:=Column;
  while J>1 do
   begin
    J:=J-1;
    if Match(Test,I,J) then
    begin
     Test:=(Test mod Sn) + 1;
     changed:=true;
    end;
   end;
 end;
end;

function RuleOne(var Per_array: Parray; Row: integer):boolean;

   var Check : boolean;
       I, J : integer;

   begin
      Check := true;
      J := 1;
      while J <= (Sn-1) do
         begin
           I := J + 1;
           if Per_array[Row, J] > Per_array[Row, I] then Check := false;
           J := J + 2;
         end;
      if Check then RuleOne:= true
         else RuleOne := false;
   end;
```

```
function RuleTwo(var Per_array : Parray; Row : integer):boolean;

   var Check : boolean;
       I , J : integer;

   begin
      Check := true;
      J := 1;
      while J < (Sn-2) do
        begin
           I := J + 2;
           if Per_array[Row, J] > Per_array[Row, I]
                                          then Check := false;
           J := J + 2;
        end;
      if Check then RuleTwo := true
        else  RuleTwo := false;
   end;

function RuleThree(var Per_array: Parray; Row: integer): boolean;

  var I, J : integer;
      OK : boolean;

  begin
     I := 1 ;
     OK := true;
     while i < Sn do
       begin
         J := I + 1;
         if Per_array[Row, I] <> Per_array[Row, J] - Num_Col then
          if Per_array[Row, I] <> Per_array[Row, J] - 1 then OK := false
          else if Per_array[Row, I] mod Num_Col = 0 then OK := false;
         I := I + 2;
       end;
     RuleThree := OK;
   end;

procedure Zeroes (var Per_array : Parray);

  var Row , Column : integer;

  begin
     for Row := 1 to Snf do
       if not(RuleOne(Per_array, Row) and RuleTwo(Per_array, Row)
              and RuleThree(Per_array, Row)) then
         for Column := 1 to Sn do   Per_array [Row , Column] := 0;
  end;

procedure Pf(var Per_array : Parray; var J: integer);

  var I: integer;

  begin
     j := 0;
     for i:= 1 to Snf do
       if Per_array[I, 1] > 0 then J := J + 1;
     writeln(J);
  end;
```

```
begin
 for Column:=1 to Sn do
 begin
  Row:=1;
  Increment:=0;
  Cycle:= Factorial(Sn-Column);
  Length:= Factorial(Sn-Column+1);
  while Row < (Snf+1) do
  begin
   if ((Row mod Cycle) = 1) or (Cycle=1) then Increment:=(Increment mod Sn) +
   if (Row mod Length) = 1 then Increment:= 1;
   Look_write(Increment,Row,Column);
   for Counter:= 1 to Cycle do
    Per_array[Row+Counter-1,Column] := Increment;
   Row:=Row+Cycle;
  end;
 end;
 Zeroes(Per_array);
 Pf(Per_array, J);
 for Row:= 1 to Snf do
 begin
  if Per_array[Row, 1] <> 0 then
  begin
   for Column:= 1 to Sn do
    write(lst, Per_array[Row,Column],' ');
    writeln(lst );
  end;
 end;
end.
```