TILING SQUARES

ERWIN KALVELAGEN

Abstract. Tiling squares is a difficult geometric problem. In this document we illustrate how small instances of the problem can be solved using Mixed Integer Programming. Larger instances, however, are beyond the reach of this method.

1. Introduction

In this section we deal with the problem of tiling (integer) squares. We try to fill a given $n \times n$ square with smaller $k \times k$ squares. An example is given in figure 1 (see [1]).

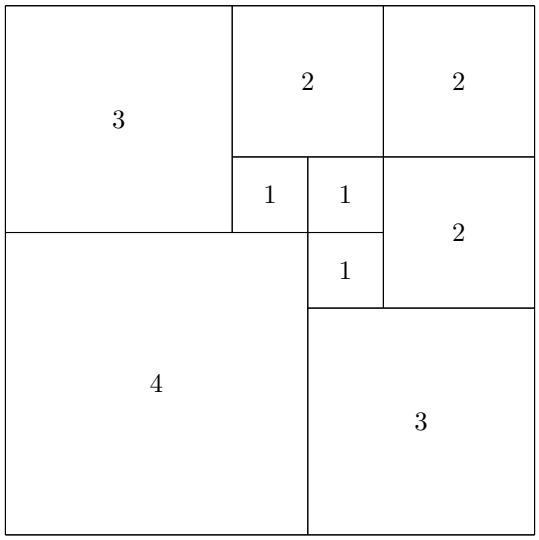

Figure 1. Tiling a 7×7 square

2. Modeling

Once we have a set of tiles that can fill the square, positioning them can be modeled as follows. For each tile i and j we need to make sure they don't overlap. Let's assume the following notation: x_i and y_i are the (x, y) coordinates of the

Date: September 22, 2002.

lower-left corner of each tile and s_i is the length of the side. At least one of the following constraints must hold:

$$x_i \geq x_j + s_j \tag{1}$$

$$x_i + s_i \leq x_j \tag{2}$$

$$y_i \geq y_j + s_j \tag{3}$$

$$y_i + s_i \leq y_j \tag{4}$$

The following big-M formulation can be used to implement this:

$$x_i \geq x_j + s_j - M\delta_{1,i,j} \tag{5}$$

$$x_i + s_i \leq x_j + M\delta_{2,i,j} \tag{6}$$

$$y_i \geq y_j + s_j - M\delta_{3,i,j} \tag{7}$$

$$y_i + s_i \leq y_j + M\delta_{4,i,j} \tag{8}$$

$$\sum_k \delta_{k,i,j} \leq 3 \tag{9}$$

with $\delta_{k,i,j}$ binary variables. Choosing M as two times the side of the total square is large enough. Using symmetry we only need to use the equations for $i > j$. The complete model is:

2.0.1. *Model square7.gms.*

```
$title Tiling the 7 x 7 square
$ontext

    Find the location of the tiles so that they don't overlap.

    Erwin Kalvelagen, June 2001

$offtext

sets
   i 'pool of tiles' /i1*i9/
   c 'non-overlap directions' /c1*c4/
;

alias (i,j);

set lt(i,j) 'less-then';
lt(i,j)$(ord(i) > ord(j)) = yes;

scalar size 'size of outer square ' /7/;

parameter s(i) 'side of tile i' /
  i1         4
  (i2,i3)    3
  (i4*i6)    2
  (i7*i9)    1
/;

parameter surf(i) 'surface of tile i';
surf(i) = sqr(s(i));

abort$(sum(i,surf(i))<>sqr(size)) "area can not be covered";

integer variable
   x(i) 'x coordinate of square i'
   y(i) 'y coordinate of square i'
;
binary variable
   d(c,i,j)  'overlap detection'
;
variable z;
```

```
equation
   overlap1(i,j) 'prevent overlap'
   overlap2(i,j) 'prevent overlap'
   overlap3(i,j) 'prevent overlap'
   overlap4(i,j) 'prevent overlap'
   countd(i,j)   'prevent overlap'
   dummy
;

*
* choose M as small as possible
*
scalar M 'big-M';
M = 2 * size;

*
* one of these must hold for any (i,j)
*
overlap1(lt(i,j)).. x(i)        =g= x(j) + s(j) - M * d('c1',i,j);
overlap2(lt(i,j)).. x(i) + s(i) =l= x(j)        + M * d('c2',i,j);
overlap3(lt(i,j)).. y(i)        =g= y(j) + s(j) - M * d('c3',i,j);
overlap4(lt(i,j)).. y(i) + s(i) =l= y(j)        + M * d('c4',i,j);

*
* counting constraint
*
countd(lt(i,j)).. sum(c, d(c,i,j)) =l= 3;

*
* all squares within major square
*
x.lo(i)=0;
x.up(i) = size - s(i);
y.lo(i)=0;
y.up(i) = size - s(i);

*
* dummy objective
*
dummy.. z=e=0;

model squares /all/;

solve squares minimizing z using mip;
```

A more difficult problem arises if we want to let the model determine the tiles. As an example we consider the following sequence. Let

$$h(n) = \text{the smallest multiplicity of any tile needed} \tag{10}$$

The following numbers $h(n), n \geq 2$ are listed in [2]: 4, 5, 4, 4, 4, 3, 4, 3, 4, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2. I.e. $h(9) = 3$, which indicates that we can form a square of 9×9 if tiles can appear up to three times. To prove that we can form such a square we create a pool of tiles $i = 1..3(n-1)$ with side $s_i = (1, 1, 1, 2, 2, 2, 3, 3, 3, ..., 8, 8, 8)$, and introduce binary variables u_i defined by:

$$u_i = \begin{cases} 1 & \text{if tile } i \text{ is used} \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

The counting constraint on $\delta_{k,i,j}$ (9) need to be changed into something like:

$$\sum_k \delta_{k,i,j} \le 3 + (1-u_i) + (1-u_j) \tag{12}$$

to make sure that the non-overlap constraints are not used on any (i,j) for which not both tile i and tile j are used. The additional constraint that is now needed is:

$$\sum_i s_i^2 u_i = n^2 \tag{13}$$

As a refinement we consider to specify priorities on the integer variables. Clearly the u_i variables should be dealt with before the other ones. Within the u_i larger tiles should be considered first. Prioritizing x and δ is more difficult, but it makes some sense to start with larger tiles.

A second improvement is to add constraints:

$$\delta_{k,i,j} \ge 1 - u_i \tag{14}$$

$$\delta_{k,i,j} \ge 1 - u_j \tag{15}$$

which forces $\delta_{k,i,j} = 1$ if $u_i = 0$ or $u_j = 0$.

The complete model is listed below:

2.0.2. *Model square9.gms.*

```
$title Tiling squares
$ontext

    Find a configuration of tiling squares, with multiplicity of 3,
    such that a 9 x 9 square filled by them is completely covered.

    Erwin Kalvelagen, June 2001

$offtext

sets
   i 'pool of tiles' /i1*i24/
   c 'non-overlap directions'  /c1*c4/
;

alias (i,j);

set lt(i,j) 'less-then';
lt(i,j)$(ord(i) > ord(j)) = yes;

scalar size 'size of outer square ' /9/;

parameter s(i) 'side of tile i';
s(i) = ceil(ord(i)/3);
display s;

parameter surf(i) 'surface of tile i';
surf(i) = sqr(s(i));

integer variable
   x(i) 'x coordinate of tile i'
   y(i) 'y coordinate of tile i'
;
binary variable
   u(i)      'square i is used'
   d(c,i,j)  'overlap detection'
;
variable z;

equation
   overlap1(i,j) 'prevent overlap'
```

```
   overlap2(i,j) 'prevent overlap'
   overlap3(i,j) 'prevent overlap'
   overlap4(i,j) 'prevent overlap'
   countd(i,j)   'prevent overlap'
   area          'square must be covered'
   extra1(c,i,j) 'makes formulation tighter'
   extra2(c,i,j) 'makes formulation tighter'
;

scalar M 'big-M';
M = 2 * size;

*
* one of these must hold for any (i,j)
*
overlap1(lt(i,j)).. x(i)        =g= x(j) + s(j) - M * d('c1',i,j);
overlap2(lt(i,j)).. x(i) + s(i) =l= x(j)        + M * d('c2',i,j);
overlap3(lt(i,j)).. y(i)        =g= y(j) + s(j) - M * d('c3',i,j);
overlap4(lt(i,j)).. y(i) + s(i) =l= y(j)        + M * d('c4',i,j);

*
* only if both tiles i and j are actually used
*
countd(lt(i,j)).. sum(c, d(c,i,j)) =l= 3 + (1-u(i)) + (1-u(j));

*
* if u(i) = 0 or u(j) = 0 then set d(c,i,j) to 1
* not needed but may help
*
extra1(c,lt(i,j)).. d(c,i,j) =g= 1-u(i);
extra2(c,lt(i,j)).. d(c,i,j) =g= 1-u(j);

*
* area  covered
*
area.. z =e= sum(i, u(i)*surf(i));
z.fx = sqr(size);

*
* priorities
*
u.prior(i) = card(i) - ord(i);
d.prior(c,i,j) = 2*card(i)-max(ord(i),ord(j));
x.prior(i) = 3*card(i) - ord(i);
y.prior(i) = 3*card(i) - ord(i);

*
* all tiles within major square
*
x.lo(i)=0;
x.up(i) = size - s(i);
y.lo(i)=0;
y.up(i) = size - s(i);

model squares /all/;

option iterlim=1000000,reslim=10000,optcr=0;

*
* cplex option file
*
file f  /cplex.opt/;
putclose f 'mipemphasis 1';
squares.optfile=1;

squares.prioropt = 1;

option mip=cplex;

solve squares maximizing z using mip;
```

MIP models are quite unpredictable in performance. This is illustrated by the following results.

Cplex Options	Nodes used
`mipemphasis 1`	24004
`mipemphasis 1`, priorities	42938
`mipemphasis 1`, extra equations	2479
`mipemphasis 1`, priorities, extra equations	1955

TABLE 1. GAMS CPLEX Results

The use of priorities initially seems to deteriorate the performance of the model. However if we add the extra constraints to the model, which in itself has a great effect on the node count, the use of priorities seems to modestly help the solver.

The Cplex option `mipemphasis 1` tells the solver that we aim for feasible solutions instead of optimal solutions. Indeed in our model there is no optimization at all: we only want the solver to find feasible solutions. In practice these models are very difficult to solve as MIP solvers are organized around the concept of an objective that can be used to ignore parts of the branch-and-bound tree. It is often argued that for these type of models, techniques such as constraint programming are more appropriate.

3. Open problem

I have not been able to proof yet that $h(110) = 1$ using the above techniques.

4. Thanks

This problem was suggested to me by Paul van der Eijk, who also came up with the idea to use a pool of tiles and corresponding usage variables u_i.

References

1. Erich Friedma, *Integer square tilings, problem of the month december 1998*, `http://www.stetson.edu/~efriedma/mathmagic/1298.html`, 1998.
2. N. J. A. Sloane, *The on-line encyclopedia of integer sequences, sequence a036444*, `http://www.research.att.com/~njas/sequences/Seis.html`, 2001.

GAMS Development Corp., Washington DC
E-mail address: `erwin@gams.com`