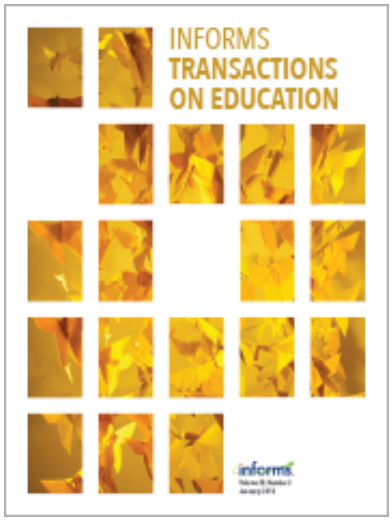


INFORMS Transactions on Education

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

The Gunport Problem

Martin J. Chlond, Robert Bosch,

To cite this article:
Martin J. Chlond, Robert Bosch, (2006) The Gunport Problem. INFORMS Transactions on Education 6(2):37-43.
https://doi.org/10.1287/ited.6.2.37

The Gunport Problem

Martin J.Chlond
University of Central Lancashire
Preston, PR1 2HE UK
mchlond@uclan.ac.uk

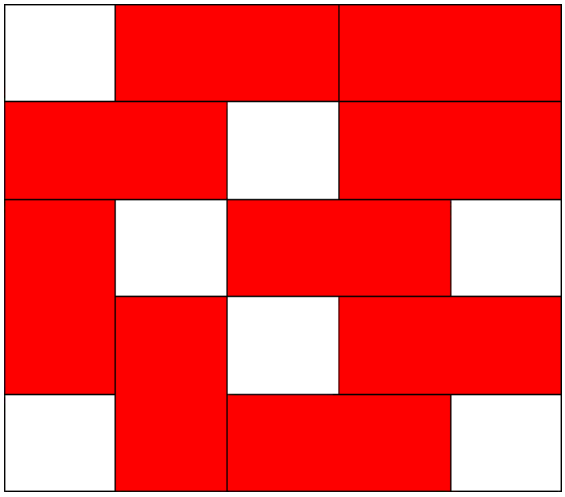
Robert Bosch
Oberlin College
Department of Mathematics
Oberlin, Ohio 4074
bobb@cs.oberlin.edu

If one places 2x1 dominoes on an mxn board (each domino covering exactly two unit squares of the board) until no more dominoes can be accommodated, one notices that there may be a number of 1x1 squares, or "holes" left vacant.

1. Introduction

The "Gunport Problem" was originated by Bill Sands (1971) and popularised by Martin Gardner in his long-running Mathematical Games column for Scientific American (April 1974). The original paper by Sands is available online [(1)] and an extract from Gardner's article is reprinted in "The Colossal Book of Short Puzzles and Problems". We introduce the problem by way of the following quote from Sands' paper.

The "holes" are analogous to ports in the sides of a warship or fortress through which cannons may be aimed. The problem is to identify arrangements of dominoes on arbitrary sized boards that maximize the number of holes left vacant or, equivalently, minimize the number of dominoes placed. The following 5x5 solution containing 7 holes is given as an example.



2. Formulation

In a first attempt at formulation we begin with a number of dominoes (p) sufficient to cover the board with at most a single cell vacant, $floor(mn/2)$ (where m = rows and n = columns). This is naive upper bound on the number of dominoes required. We define sets $M = 1..m$, $N = 1..n$, $P = 1..p$ and decision variables $x_{i,j,k} = 1$ if cell $\{i,j\}$ is occupied by domino k and variables $d_k = 1$ if domino k is used. The objective is to minimize the number of dominoes used subject to constraints that ensure that a domino, if placed, covers two adja-

(1) http://www.jstor.org/

cent cells, each cell is covered by, at most, a single domino and, finally, no two adjacent cells are vacant.

$$\text{Minimize} \quad \sum_{k \in P} d_k$$

1. Used dominoes in adjacent cells.

$$\sum_{\substack{l=j-1 \\ l \geq 1 \\ l \leq n \\ l \neq j}}^{j+1} x_{i,l,k} + \sum_{\substack{l=i-1 \\ l \geq 1 \\ l \leq m \\ l \neq i}}^{i+1} x_{l,j,k} \quad \forall i \in M, j \in N, k \in P$$

2. Domino, if placed, covers two cells.

$$\sum_{i \in M} \sum_{j \in N} x_{i,j,k} = 2d_k \quad \forall k \in P$$

3. Each cell covered by, at most, one domino.

$$\sum_{k \in P} x_{i,j,k} \leq 1 \quad \forall i \in M, j \in N$$

4. No two adjacent cells vacant (rows).

$$\sum_{c=j}^{j+1} \sum_{k \in P} x_{i,c,k} \geq 1 \quad \forall i \in M, j \in \{1..n-1\}$$

5. No two adjacent cells vacant (columns).

$$\sum_{r=i}^{i+1} \sum_{k \in P} x_{r,j,k} \geq 1 \quad \forall i \in \{1..m-1\}, j \in N$$

An OPL model of this formulation is available [(2)]. The performance of this model is wholly unimpressive with only a 4x4 grid solved in a reasonable amount of time. Inspection of the solution procedure reveals that a solution is found rather quickly but the branch-and-bound procedure continues to look for a better solution which does not, in fact, exist.

3. Formulation II

A couple of insights into the nature of solutions to Gunport problems will allow us to modify and improve the above model. Bill Sands (1971) proved that if ***mod(m,3) or mod(n,3) = 0*** then ***p = mn/3*** (where ***m*** = rows, ***n*** = columns and ***p*** = optimum number of dominoes). This may be achieved using a simple repeating pattern. Murray R. Pearce later conjectured that if ***mod(m,3) = mod(n,3)*** then ***p = (mn+2)/3*** or if ***mod(m,3) mod(n,3)*** then ***p = (mn+1)/3*** (see Gardner). We may use these insights to amend the above formulation. The objective now is to place ***p*** dominoes in accordance with the conditions of the puzzle. We no longer need the objective function or the $d_{i,j}$ variables and constraints (2) are amended to:

$$\sum_{i \in M} \sum_{j \in N} x_{i,j,k} = 2 \quad \forall k \in P$$

An OPL model of this modified formulation is available [(3)]. This improved model found solutions up to and including the 7x7 board but thereafter the processing time increases dramatically and no result was obtained for an 8x8 board after 5 hours. This was quite disappointing as a solution to an 8x10 board was found by Captain John C. Huvall in 1975 (see Gardner, 2006).

4. Formulation III

An alternative approach is to define a three dimensional array ***DC*** where $dc_{i,j,k} = 1$ if cell ***{i,j}*** is covered when a domino is in position ***k***, **0** otherwise. Decision variables $x_k = 1$ if position ***k*** adopted, **0** otherwise. We also need variables $y_{i,j} = 1$ if cell ***{i,j}*** covered, **0** otherwise. We include constraints to ensure that each of ***p*** dominoes is placed, each cell is covered by, at most, a single domino and finally, as above, no two adjacent cells are vacant. An OPL model of this re-formulation is available [(4)].

The improvement in solution times enjoyed by this approach is quite dramatic. For example, the 8x10 problem was unsolved after seven days using the modified initial formulation above whereas this re-formulation found a solution in less than four seconds.

(2) http://ite.pubs.informs.org/Vol6No2/ChlondBosch/gunport1.php
(3) http://ite.pubs.informs.org/Vol6No2/ChlondBosch/gunport2.php
(4) http://ite.pubs.informs.org/Vol6No2/ChlondBosch/gunport3.php

4. Formulation IV

A final approach considers the equivalent problem of filling a rectangle with monominoes and dominoes so that no two monominoes are adjacent and the number of monominoes is minimized (or a specified number of monominoes is placed). In this case three sets of decision variables are defined.

$h_{r,c} = 1$ if a domino is placed horizontally with its left-most square covering square *{r,c}*

$v_{r,c} = 1$ if a domino is placed vertically with its topmost square covering square *{r,c}*

$m_{r,c} = 1$ if a monomino is placed in square *{r,c}*

One set of constraints ensures that each square is covered exactly once (by a horizontally placed domino, a vertically placed domino, or a monomino).

Another set of constraints prevents us from putting two monominoes next to each other.

Two additional sets of constraints eliminate symmetries. They make sure that the bottom half of the board contains at least at least as many monominoes as the top, and the right half contains at least as many as the left half. These symmetry constraints are important in that they have a marked effect on solution times. An OPL implementation of this approach is available [5].

Experimentation with this final approach demonstrates its superiority over the previous method. For moderate sized boards the improvement is already noticeable but as the board size increases the differences in solution times rapidly become much more pronounced.

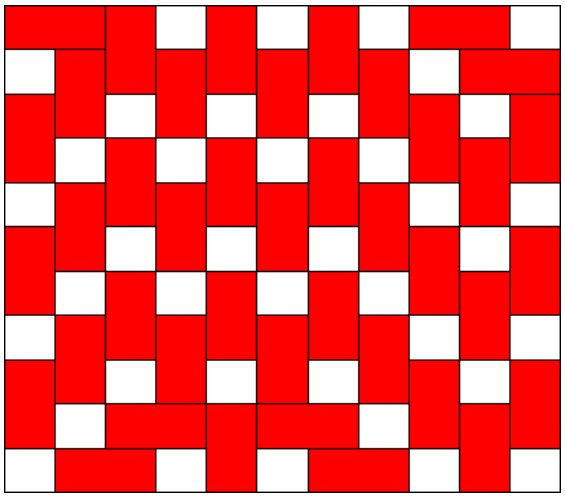
5. Comparison of methods

The following table of solution times for a range of board sizes allows for a direct comparison of the four methods.

Board size	Method 1	Method 2	Method 3	Method 4
4x4	81 secs	0.65 secs	0.21 secs	
5x5	> 5 hrs	0.76 secs	0.21 secs	
6x6		67 secs	0.76 secs	
7x7		41 secs	0.87 secs	
8x8		> 24 hrs	1.31 secs	0.32 secs
9x9			1.90 secs	1.31 secs
10x10			4.04 secs	1.70 secs
11x11			114 secs	6.89 secs
12x12			> 24 hrs	14.1 mins

The marked differences in solution times witnessed by the above table provide an emphatic demonstration of the benefits that may be gained by the re-formulation of difficult problems.

A solution for board size 11x11 with 39 holes is as follows.



6. Final note

An interesting modification of the Gunport problem may be found at Tim's Interactive Puzzle [6] . An additional requirement is that no domino can slide in any direction. Further constraints may be added to the above models to enforce this condition and this is left as an exercise for the reader.

References

Gardner, M. (2006), *The Colossal Book of Short Puzzles and Problems*, W.W. Norton & Company.

Sands, B. (1971), *The Gunport Problem*, Mathematics Magazine, Vol. 44, pp. 193-196

[5] http://ite.pubs.informs.org/Vol6No2/ChlondBosch/gunport4.php
[6] http://sakharov.net/puzzle

Appendix

gunport1.mod

```
/*
 Model name   : gunport1.mod
 Description   : solves Gunport puzzles - modified
 Source        : Martin Gardner
 Date written  : 3/2/06
 Written by    : Martin Chlond, Lancashire Business School
 Email         : mchlond@uclan.ac.uk
*/

int m = 4;
int n = 4;
int p = 8;

range M = 1..m;
range N = 1..n;
range P = 1..p;

dvar boolean x[M,N,P];
dvar boolean d[P];

minimize sum(k in P) d[k];

subject to {

 /* each domino in adjacent cells */
 forall(i in M,j in N,k in P)
     sum(l in j-1..j+1 : l >= 1 && l <= n && l != j) x[i,l,k]+
     sum(l in i-1..i+1 : l >= 1 && l <= m && l != i) x[l,j,k] >= x[i,j,k];

 /* each domino covers two cells */
 forall(k in P) sum(i in M,j in N) x[i,j,k] == 2*d[k];

 /* each cell covered by, at most, one domino */
 forall(i in M,j in N) sum(k in P) x[i,j,k] <= 1;

 /* no two adjacent cells vacant (rows) */
 forall(i in M,j in 1..n-1) sum(c in j..j+1,k in P) x[i,c,k] >= 1;

 /* no two adjacent cells vacant (columns) */
 forall(i in 1..m-1,j in N) sum(r in i..i+1,k in P) x[r,j,k] >= 1;

}
execute {

 /* format and display output */
 for(i in M) {
  for(j in N) {
   rt = 0;
   for(k in P) {
    rt = rt+k*x[i][j][k];
   }
   write(rt);
   write(" ");
  }
  writeln();
 }

}
```

gunport2.mod

```
/*
 Model name   : gunport2.mod
 Description  : solves Gunport puzzles
 Source       : Martin Gardner
 Date written : 3/2/06
 Written by   : Martin Chlond, Lancashire Business School
 Email        : mchlond@uclan.ac.uk
*/

int m = 7;
int n = 7;
int p;

int m3 = m mod 3;
int n3 = n mod 3;

execute {

  /* compute required number of dominoes */
  if (m3*n3==0)
    p = m*n/3;
    else
      if (m3==n3)
        p = (m*n+2)/3;
        else
          p = (m*n+1)/3;
}

range M = 1..m;
range N = 1..n;
range P = 1..p;

dvar boolean x[M,N,P];

subject to {

  /* each domino in adjacent cells */
  forall(i in M,j in N,k in P)
        sum(l in j-1..j+1 : l >= 1 && l <= n && l != j) x[i,l,k]+
        sum(l in i-1..i+1 : l >= 1 && l <= m && l != i) x[l,j,k] >= x[i,j,k];
  /* each domino covers two cells */
  forall(k in P) sum(i in M,j in N) x[i,j,k] == 2;

  /* each cell covered by, at most, one domino */
  forall(i in M,j in N) sum(k in P) x[i,j,k] <= 1;

  /* no two adjacent cells vacant (rows) */
  forall(i in M,j in 1..n-1) sum(c in j..j+1,k in P) x[i,c,k] >= 1;

  /* no two adjacent cells vacant (columns) */
  forall(i in 1..m-1,j in N) sum(r in i..i+1,k in P) x[r,j,k] >= 1;

}

execute {

  /* format and display output */
  for(i in M) {
    for(j in N) {
      rt = 0;
      for(k in P) {
        rt = rt+k*x[i][j][k];
      }
      write(rt);
      write(" ");
    }
    writeln();
  }

}
```

gunport3.mod

```
/*
 Model name   : gunport3.mod
 Description  : solves Gunport puzzles - re-formulation
 Source       : Martin Gardner
 Date written : 23/3/06
 Written by   : Martin Chlond, Lancashire Business School
 Email        : mchlond@uclan.ac.uk
*/

int m = 8;
int n = 10;
int p;

int npos = 2*m*n-n-m;

range NP = 1..npos;
range M = 1..m;
range M1 = 1..m-1;
range N = 1..n;
range N1 = 1..n-1;

int dc[M,N,NP];

int m3 = m mod 3;
int n3 = n mod 3;

execute {

 /* compute required number of dominoes */
 if (m3*n3==0)
   p = m*n/3;
   else
     if (m3==n3)
       p = (m*n+2)/3;
       else
         p = (m*n+1)/3;

 /* identify domino configurations */
 for(j in N)
  for(i in M1) {
    k = i+(m-1)*(j-1);
    dc[i][j][k] = 1;
    dc[i+1][j][k] = 1;
  }
 for(i in M)
  for(j in N1) {
    k = j+(n-1)*(i-1)+n*(m-1);
    dc[i][j][k] = 1;
    dc[i][j+1][k] = 1;
   }
}

dvar boolean x[NP];
dvar int+ y[M,N];

subject to {

 /* correct number of dominoes used */
 sum(k in NP) x[k] == p;

 /* count number of dominos covering each cell */
 forall(i in M,j in N) sum(k in NP) dc[i,j,k]*x[k] == y[i,j];

 /* no more than one domino covering each cell */
 forall(i in M,j in N) y[i,j] <= 1;

 /* no two adjacent cells vacant (rows) */
 forall(i in M,j in 1..n-1) sum(c in j..j+1) y[i,c] >= 1;

 /* no two adjacent cells vacant (columns) */
 forall(i in 1..m-1,j in N) sum(r in i..i+1) y[r,j] >= 1;

}
execute {

 /* format and display output */
 for(i in M) {

  for(j in N) {
    rt = 0;
    ct = 1;
    for(k in NP) {

      rt = rt+ct*dc[i][j][k]*x[k];
      if(x[k]==1) ct=ct+1;
    }
    write(rt);
    write(" ");
  }
  writeln();
 }
```

gunport4.mod

```
/*
 Model name   : gunport4.mod
 Description  : solves Gunport puzzles
 Source       : Martin Gardner
 Date written : 28/3/06
 Written by   : Robert Bosch, Oberlin College
 Email        : bobb@cs.oberlin.edu
*/

int nbRows = 7;
int nbCols = 9;
int p;

dvar boolean h[1..nbRows,1..nbCols-1];
dvar boolean v[1..nbRows-1,1..nbCols];
dvar boolean m[1..nbRows,1..nbCols];

int m3 = nbRows mod 3;
int n3 = nbCols mod 3;

int fhr;
int shr;
int fhc;
int shc;

execute {
  /* compute required number of monominoes */
  if (m3*n3==0)
    p = nbRows*nbCols/3;
    else
      if (m3==n3)
        p = (nbRows*nbCols-4)/3;
        else
          p = (nbRows*nbCols-2)/3;

  /* compute ranges for symmetry constraints */
  fhr = Math.floor(nbRows/2);
  if (nbRows % 2 == 0)
    shr = fhr+1;
  else
    shr = fhr+2;
  fhc = Math.floor(nbCols/2);
  if (nbCols % 2 == 0)
    shc = fhc+1;
  else
    shc = fhc+2;
}
maximize sum(r in 1..nbRows, c in 1..nbCols) m[r,c];

subject to {

  /* Interior cells */
  forall (r in 2..nbRows-1, c in 2..nbCols-1)
    h[r,c-1] + h[r,c] + v[r-1,c] + v[r,c] + m[r,c] == 1;
  /* Edge cells, top row */
  forall (c in 2..nbCols-1)
    h[1,c-1] + h[1,c] + v[1,c] + m[1,c] == 1;
  /* Edge cells, bottom row */
  forall (c in 2..nbCols-1)
    h[nbRows,c-1] + h[nbRows,c] + v[nbRows-1,c] + m[nbRows,c] == 1;
  /* Edge cells, leftmost column */
  forall (r in 2..nbRows-1)
    h[r,1] + v[r-1,1] + v[r,1] + m[r,1] == 1;
  /* Edge cells, rightmost column */
  forall (r in 2..nbRows-1)
    h[r,nbCols-1] + v[r-1,nbCols] + v[r,nbCols] + m[r,nbCols] == 1;
  /* Cell (1,1) */
  h[1,1] + v[1,1] + m[1,1] == 1;
  /* Cell (1,nbCols) */
  h[1,nbCols-1] + v[1,nbCols] + m[1,nbCols] == 1;
  /* Cell (nbRows,1) */
  h[nbRows,1] + v[nbRows-1,1] + m[nbRows,1] == 1;
  /* Cell (nbRows,nbCols) */
  h[nbRows,nbCols-1] + v[nbRows-1,nbCols] + m[nbRows,nbCols] == 1;

  /* No horizontally adjacent monominoes */
  forall (r in 1..nbRows, c in 1..nbCols-1)
    m[r,c] + m[r,c+1] <= 1;
  /* No vertically adjacent monominoes */
  forall (r in 1..nbRows-1, c in 1..nbCols)
    m[r,c] + m[r+1,c] <= 1;

  /* eliminate symmetries */
  sum( r in 1..fhr, c in 1..nbCols) m[r,c]
    <= sum( r in shr..nbRows, c in 1..nbCols) m[r,c];
  sum( r in 1..nbRows, c in 1..fhc) m[r,c]
    <= sum( r in 1..nbRows, c in shc..nbCols) m[r,c];

  /* place specified number of numbers */
  sum(r in 1..nbRows, c in 1..nbCols) m[r,c] == p;

}
```