PaintByNumbers.mws

Paint By Numbers

31 March 2001

We can generalize the technique of interacting copies to write a single procedure to handle a whole class of puzzles. These puzzles are known as Paint-by-Numbers, and also as Nonograms, or O-e-kaki in Japanese. We are given a rectangular grid of pixels. For each row and column, we are told the lengths of the groups of pixels that are black in that row or column. The challenge is to use that information to figure put the exact placement of the pixels.

```
> restart;
```

This application uses a Maple package called **LP** , contained in the file "LogicProblem.mpl", which is read by the next line. Make sure that file is in the same directory as this worksheet before executing.

```
> read "LogicProblem.mpl";
```

```
> `&subset`:= proc(A,B)
local i;
for i in A do if not member(i,B) then return false fi od;
true
end proc:
```

```
> PaintByNumbers:= proc(Rowdata::list(list(nonnegint)), Coldata::list(list(nonnegint)))
local i,j,rows,cols,nfilled,BR,BC,P_BC,P_BR,ColConstraints,RowConstraints,colors
,P,row,col,blanks,filled,fills,blank,Types,Type,changes
,TypeToColor, NotColor, k, k1, k2, k3, t, Interpret, Coordinate, ShowIter
,IncreaseInfo, PrintSoln, st;

IncreaseInfo:= proc(M, opts)
use `&?`= M:-`&?` in
if [&? Stats][6] = opts[1] then :-infolevel[all]:= opts[2]; return false, true fi
end use;
false, false
end proc;

Interpret:= proc(M)
local h, Type, Q, C, Nots;
C:= [];
# Note that 'colors' is identical, both internally and externally, in both problems. Also note that P_BR and P_BC are
# equivalent as sets; they are merely ordered differently in the two copies of the problem.
for h in P_BC do
use `&?`= M:-`&?` in Q:= &? h end use;
if Q::`=` then C:= [op(C), Distinct([h, NotColor[attributes(op([2,1], Q))]])]
else
Nots:= op(2, Q);
for Type in [blanks, fills] do if Type &subset Nots then C:= [op(C), Distinct([h, Type])]; break fi od
fi
od;
C
end proc;

Coordinate:= proc(This, That)
local changes;
# The "Interpretation" process is quite expensive in the early stages, when there are few equivalences and many antiequivalences.
# We only want to do it if there is nothing else that can be
# concluded in the Master under the current guess. Thus, we check "Changes".
use `&?`= This:-`&?` in changes:= &? Changes end use;
if changes = :-sav_changes then return false, false else :-sav_changes:= changes fi;
```

```
use `&?`= That[1]:-`&?`, Satisfy= That[1]:-Satisfy, GoBack= That[1]:-GoBack, Guess= That[1]:-Guess in
if &? CountGuesses = 1 then GoBack() fi;
Guess(Dummy(proc() false, true end, []));
Satisfy(Interpret(This));
evalb(&? CountGuesses = 0), This:-IsComplete(), `if`(&? Changes > 0, Interpret(That[1]), NULL)
end use;
end;

PrintSoln:= proc(Sol, rows, cols)
local colors, i, j;
colors:=
rtable
(1..rows, 1..cols
,proc(i,j)
local item;
item:= attributes(Sol[cols*(i-1)+j, 2]);
`if`(item=filled, [.25,.25,.4], `if`(item=blank, [1,1,.85], [.5,.5,.5]))
end
);
PLOT
(POLYGONS
(seq(seq([[j,-i], [j,-i-1], [j+1,-i-1], [j+1,-i]], i= 1..rows), j= 1..cols)
,COLOUR(RGB, seq(seq(op(colors[i,j]), i= 1..rows), j= 1..cols))
), `plot/options2d`(scaling= constrained, axes= none)
)
end proc;

ShowIter:= proc(M, dims)
local S;
use `&?`= M:-`&?` in print(PrintSoln(P_BR &? ``, dims[1], dims[2])) end use;
:-frame:= :-frame+1;
:-Frames[:-frame]:= eval(S);
false, false
end proc;

rows:= nops(Rowdata);
cols:= nops(Coldata);
nfilled:= `+`(op((map(`+`@op, Rowdata))));
if `+`(op((map(`+`@op, Coldata)))) <> nfilled then
error `Number of filled squares by rows and by columns is inconsistent.`
fi;
for i to rows do
row:= Rowdata[i];
if `+`(op(row)) + nops(row) - 1 > cols then error `A row is too long.` fi
od;
for j to rows do
col:= Coldata[j];
if `+`(op(col)) + nops(col) - 1 > rows then error `A column is too long.` fi
od;
P:= proc(i,j) option inline; cat(H,i,_,j) end;
P_BR:= [seq(seq(P(i,j), j= 1..cols), i= 1..rows)];
P_BC:= [seq(seq(P(i,j), i= 1..rows), j= 1..cols)];
blanks:= {b||(1..rows*cols-nfilled)};
fills:= {f||(1..nfilled)};
colors:= [op(blanks), op(fills)];
Types:= [blank,filled];
TypeToColor:= table([blank= blanks, filled= fills]);
NotColor:= table([seq(t= {op(colors)} minus TypeToColor[t], t= Types)]);
for t in Types do for i in TypeToColor[t] do setattribute(i, t) od od;
interface(rtablesize= rows*cols+1);
BR:= LogicProblem([P_BR, colors]);
BC:= LogicProblem([P_BC, colors]);
```

```
BC:-AutoGuess:= false;
BC:-Quiet:= true;
BC:-UniquenessProof:= false;
BR:-CollectStats:= true;
BC:-CollectStats:= true;
:-infolevel['sendmail']:= 0;

RowConstraints:= [];
k1:= 0; k2:= 0;
for i to rows do
row:= Rowdata[i];
nfilled:= `+`(op(row));
RowConstraints:=
[op(RowConstraints)
,Less([`if`(i=1, NULL, P_BR[cols*(i-1)]), f||(k1+1..k1+nfilled), `if`(i=rows, NULL, P_BR[cols*i+1])], 1)
,Less([`if`(i=1, NULL, P_BR[cols*(i-1)]), b||(k2+1..k2+cols-nfilled), `if`(i=rows, NULL, P_BR[cols*i+1])], 1)
];
k2:= k2+cols-nfilled;
k3:= k1+1;
for j to nops(row) do
for k from k3 to k3+row[j]-2 do RowConstraints:= [op(RowConstraints), Succ(f||(k+1), f||k, 1)] od;
k3:= k3+row[j];
if j<>nops(row) then RowConstraints:= [op(RowConstraints), Rel(BR:-Separated, f||(k3-1..k3), 1, [1])] fi
od;
k1:= k1+nfilled;
od;

ColConstraints:= [];
k1:= 0; k2:= 0;
for i to cols do
col:= Coldata[i];
nfilled:= `+`(op(col));
ColConstraints:=
[op(ColConstraints)
,Less([`if`(i=1, NULL, P_BC[rows*(i-1)]), f||(k1+1..k1+nfilled), `if`(i=cols, NULL, P_BC[rows*i+1])], 1)
,Less([`if`(i=1, NULL, P_BC[rows*(i-1)]), b||(k2+1..k2+rows-nfilled), `if`(i=cols, NULL, P_BC[rows*i+1])], 1)
];
k2:= k2+cols-nfilled;
k3:= k1+1;
for j to nops(col) do
for k from k3 to k3+col[j]-2 do ColConstraints:= [op(ColConstraints), Succ(f||(k+1), f||k, 1)] od;
k3:= k3+col[j];
if j<>nops(col) then ColConstraints:= [op(ColConstraints), Rel(BR:-Separated, f||(k3-1..k3), 1, [1])] fi
od;
k1:= k1+nfilled;
od;

BC:-Satisfy(ColConstraints);
use `&?`= BC:-`&?` in print(` Columnwise initial knowledge` = &? Changes) end use;
print(PrintSoln
(BR:-Satisfy
([op(RowConstraints)
,op(Interpret(BC))
,Proc(Coordinate, [BC])
,Dummy(proc() :-sav_changes:= 0; :-frame:= 0; false, true end, [])
,Dummy(ShowIter, [rows, cols])
])
,rows
,cols
));
use `&?`= BC:-`&?` in print(&? ShowStats) end use;
use `&?`= BR:-`&?` in print(&? ShowStats) end use
```

end proc:

> **infolevel[all]:= 0;**

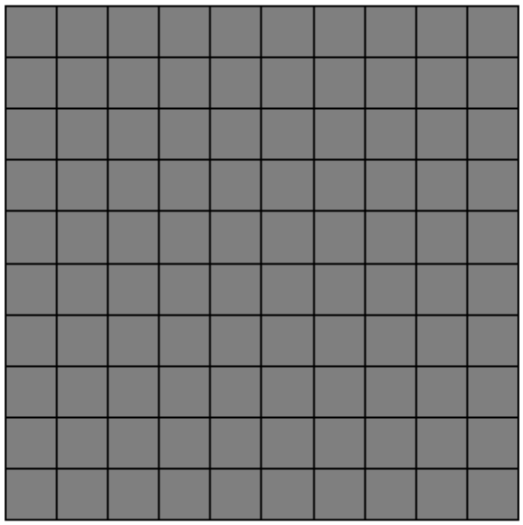
$$infolevel_{all} := 0$$

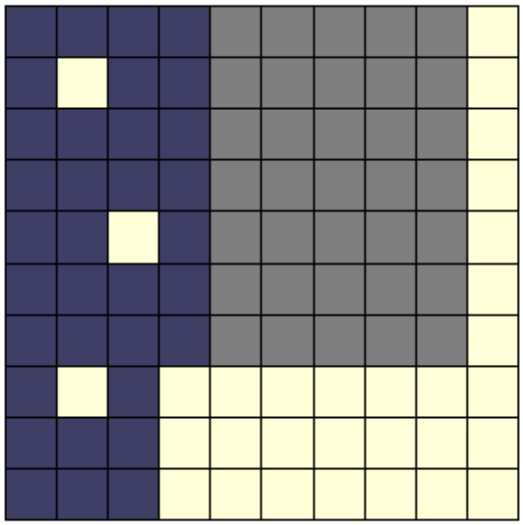
> **st:= time();**

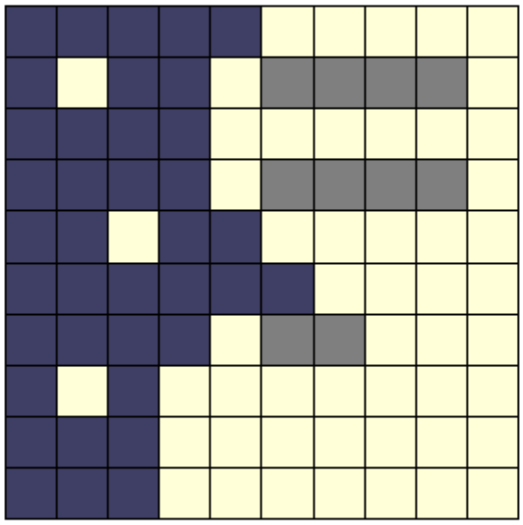
$$st := .501$$

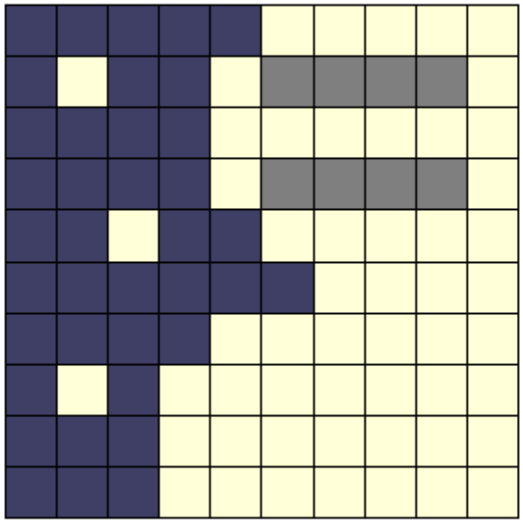
> **PaintByNumbers([[5],[1,2,1,1],[4],[4,3],[2,2],[6],[4],[1,1],[3],[3]], [[10],[1,5,2],[4,5],[7],[1,2],[1],[1,1],[1],[1,1],[0]]):**

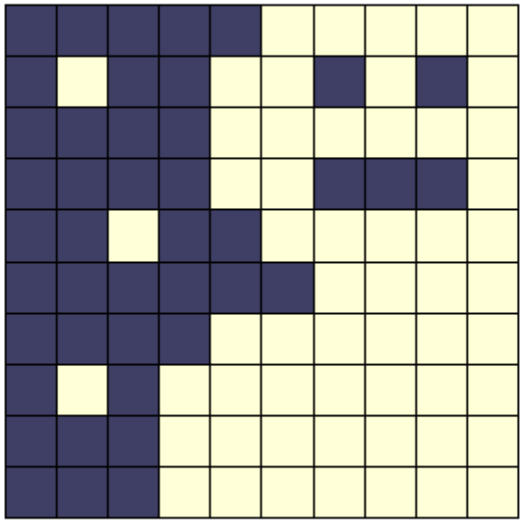
Columnwise initial knowledge = 9758



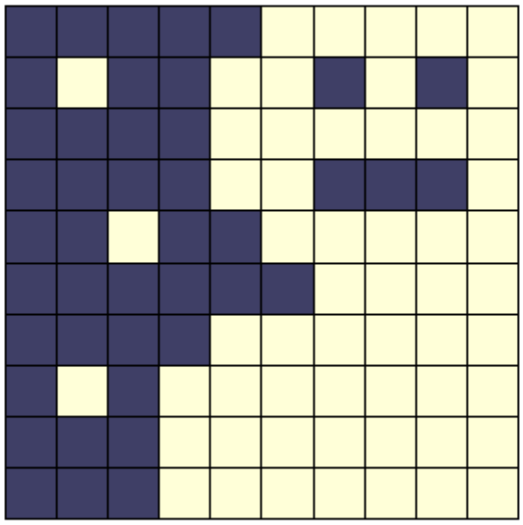








Unique solution:



$Sets = 0, Unsets = 9047, RuleOuts = 522, Elims = 19405, Transfers = 0, Pivots = 0, MaxLevel = 327, Guesses = 4, MaxDepth = 1$

$Sets = 0, Unsets = 8281, RuleOuts = 214, Elims = 18181, Transfers = 0, Pivots = 0, MaxLevel = 221, Guesses = 0, MaxDepth = 0$

> **time()-st;**

27.861