Effective Problem Solving using SAT Solvers

Curtis Bright, Jürgen Gerhard, Ilias Kotsireas, Vijay Ganesh

“. . . it is a constant source of annoyance when you come up with a clever
special algorithm which then gets beaten by translation to SAT.”

—Chris Jefferson

Introduction

SAT solvers are programs that solve the *Boolean satisfiability problem* from Boolean logic.

Maple has a built-in SAT solver that can be called using the **Satisfy** command of the **Logic** package.

```
with(Logic);
```

[*&and*, *&iff*, *&implies*, *&nand*, *&nor*, *&not*, *&or*, *&xor*, *BooleanSimplify*, *Canonicalize*, *Complement*, *Contradiction*, *Convert*, *Dual*, *Environment*, *Equivalent*, *Export*, *Implies*, *Import*, *Normalize*, *Parity*, *Random*, *Satisfiable*, *Satisfy*, *Tautology*, *TruthTable*, *Tseitin*]

```
x &and y;
```

$$x \wedge y$$

```
F := &not(x) &implies (y &or z);
```

$$F := (\neg x) \Rightarrow (y \vee z)$$

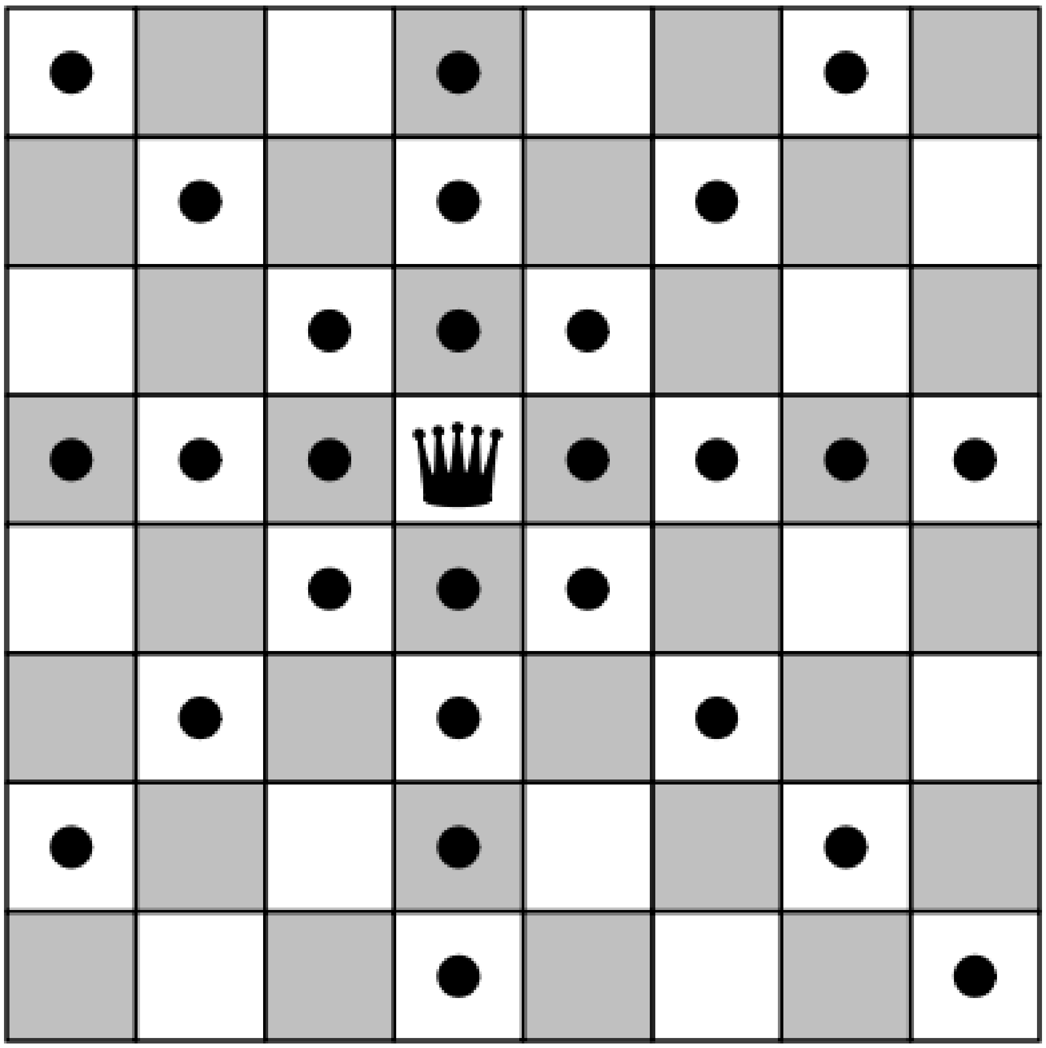
```
Satisfy(F);
```

$$\{x = \mathit{false}, y = \mathit{true}, z = \mathit{false}\}$$

In this talk we show how a variety of problems can be solved using **Satisfy**.

All of our solutions are available from the Maple Application Center.

The *n*-Queens Problem



The *n*-queens problem is to place *n* queens on an *n* by *n* chessboard such that no queens attack each other.

Variables

We use the Boolean variables $Q_{x,y}$ for x and y between 1 and n to denote if there is a queen on the square at (x, y).

```
Generate set of all variables
```

$\{Q_{1,1}, Q_{1,2}, Q_{1,3}, Q_{1,4}, Q_{1,5}, Q_{1,6}, Q_{1,7}, Q_{1,8}, Q_{2,1}, Q_{2,2}, Q_{2,3}, Q_{2,4}, Q_{2,5}, Q_{2,6}, Q_{2,7}, Q_{2,8}, Q_{3,1}, Q_{3,2}, Q_{3,3}, Q_{3,4}, Q_{3,5}, Q_{3,6}, Q_{3,7}, Q_{3,8}, Q_{4,1}, Q_{4,2}, Q_{4,3}, Q_{4,4}, Q_{4,5}, Q_{4,6}, Q_{4,7}, Q_{4,8}, Q_{5,1}, Q_{5,2}, Q_{5,3}, Q_{5,4}, Q_{5,5}, Q_{5,6}, Q_{5,7}, Q_{5,8}, Q_{6,1}, Q_{6,2}, Q_{6,3}, Q_{6,4}, Q_{6,5}, Q_{6,6}, Q_{6,7}, Q_{6,8}, Q_{7,1},$

$Q_{7,2}, Q_{7,3}, Q_{7,4}, Q_{7,5}, Q_{7,6}, Q_{7,7}, Q_{7,8}, Q_{8,1}, Q_{8,2}, Q_{8,3}, Q_{8,4}, Q_{8,5}, Q_{8,6}, Q_{8,7}, Q_{8,8}\}$

SAT encoding

Each row must contain a queen since there are *n* rows and only one queen can go in each row.

```
Generate positive constraints
```

$$Q_{1,1} \vee Q_{2,1} \vee Q_{3,1} \vee Q_{4,1} \vee Q_{5,1} \vee Q_{6,1} \vee Q_{7,1} \vee Q_{8,1}$$

If a square contains a queen then no other square in that row, column, or diagonal contains a queen.

```
Generate negative constraints
```

$$Q_{1,1} \Rightarrow (\neg (Q_{1,2} \vee Q_{1,3} \vee Q_{1,4} \vee Q_{1,5} \vee Q_{1,6} \vee Q_{1,7} \vee Q_{1,8} \vee Q_{2,1} \vee Q_{2,2} \vee Q_{3,1} \vee Q_{3,3} \vee Q_{4,1} \vee Q_{4,4} \vee Q_{5,1} \vee Q_{5,5} \vee Q_{6,1} \vee Q_{6,6} \vee Q_{7,1} \vee Q_{7,7} \vee Q_{8,1} \vee Q_{8,8}))$$

Finding a solution

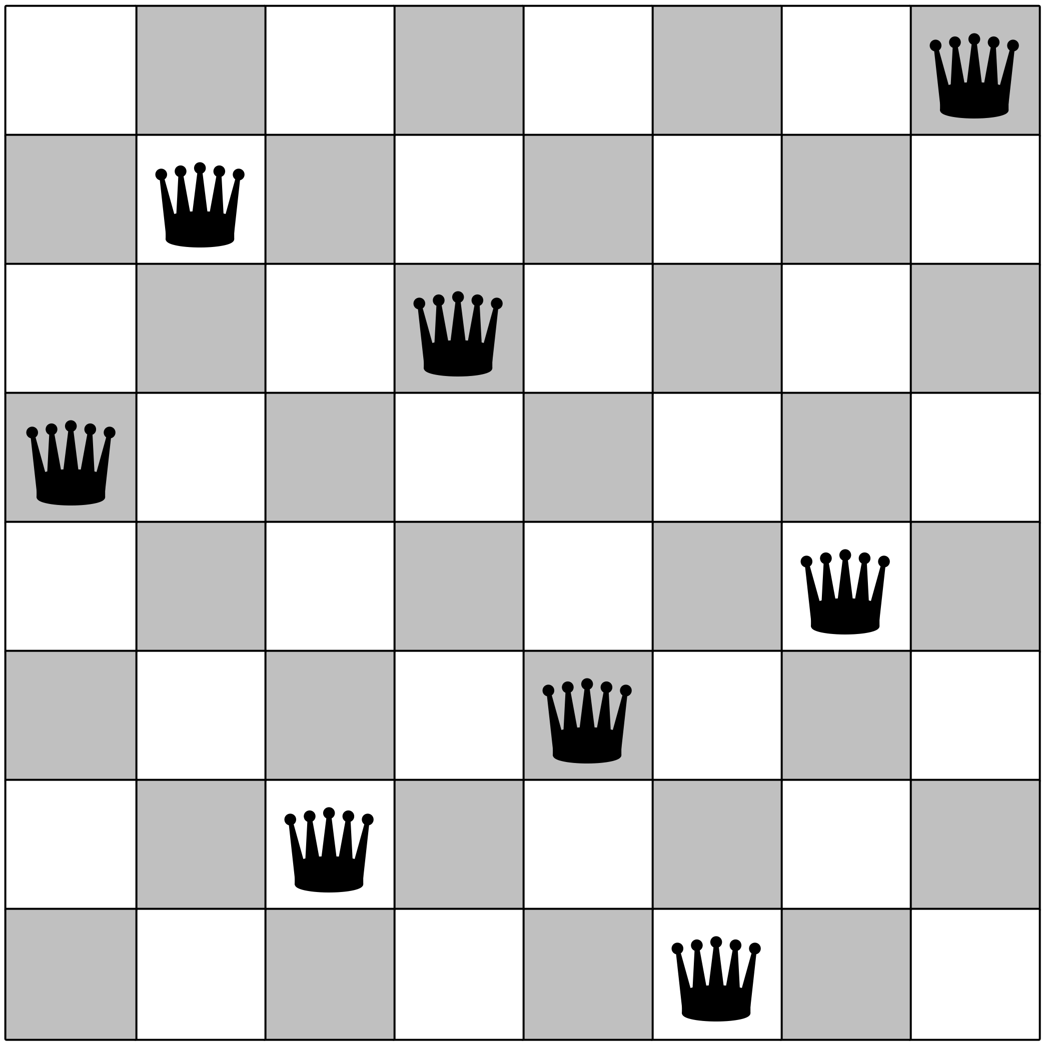
We call **Satisfy** with both the positive and negative constraints.

```
allConstraints := &and(positiveConstraints(n), negativeConstraints(n)):
satisfyingAssignment := Satisfy(allConstraints);
```

$$\mathit{satisfyingAssignment} := \{Q_{1,1} = \mathit{false}, Q_{1,2} = \mathit{false}, Q_{1,3} = \mathit{false}, Q_{1,4} = \mathit{false}, Q_{1,5} = \mathit{true}, Q_{1,6} = \mathit{false}, Q_{1,7} = \mathit{false}, Q_{1,8} = \mathit{false}, Q_{2,1} = \mathit{false}, Q_{2,2} = \mathit{false}, Q_{2,3} = \mathit{false}, Q_{2,4} = \mathit{false}, Q_{2,5} = \mathit{false}, Q_{2,6} = \mathit{false}, Q_{2,7} = \mathit{true}, Q_{2,8} = \mathit{false}, Q_{3,1} = \mathit{false}, Q_{3,2} = \mathit{true}, Q_{3,3} = \mathit{false}, Q_{3,4} = \mathit{false}, Q_{3,5} = \mathit{false}, Q_{3,6} = \mathit{false}, Q_{3,7} = \mathit{false}, Q_{3,8} = \mathit{false}, Q_{4,1} = \mathit{false}, Q_{4,2} = \mathit{false}, Q_{4,3} = \mathit{false}, Q_{4,4} = \mathit{false}, Q_{4,5} = \mathit{false}, Q_{4,6} = \mathit{true}, Q_{4,7} = \mathit{false}, Q_{4,8} = \mathit{false}, Q_{5,1} = \mathit{false}, Q_{5,2} = \mathit{false}, Q_{5,3} = \mathit{true}, Q_{5,4} = \mathit{false}, Q_{5,5} = \mathit{false}, Q_{5,6} = \mathit{false}, Q_{5,7} = \mathit{false}, Q_{5,8} = \mathit{false}, Q_{6,1}$$

$= true, Q_{6, 2} = false, Q_{6, 3} = false, Q_{6, 4} = false, Q_{6, 5} = false, Q_{6, 6} = false, Q_{6, 7} = false, Q_{6, 8}$
$= false, Q_{7, 1} = false, Q_{7, 2} = false, Q_{7, 3} = false, Q_{7, 4} = true, Q_{7, 5} = false, Q_{7, 6} = false, Q_{7, 7}$
$= false, Q_{7, 8} = false, Q_{8, 1} = false, Q_{8, 2} = false, Q_{8, 3} = false, Q_{8, 4} = false, Q_{8, 5} = false, Q_{8, 6}$
$= false, Q_{8, 7} = false, Q_{8, 8} = true\}$

```
Read satisfying assignment and draw a graphical representati
```



The Einstein Riddle

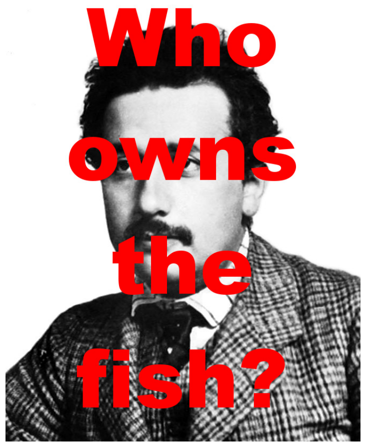
The "Einstein Riddle" is a logic puzzle apocryphally attributed to Albert Einstein.

There are five houses in a row with each house a different colour and each house owned by a man of a different nationality.

Additionally, each of the owners have a different pet, prefer a different kind of drink, and smoke a different brand of cigarette. Furthermore, we know the following information:

1. The Brit lives in the Red house.
2. The Swede keeps dogs as pets.
3. The Dane drinks tea.
4. The Green house is next to the White house, on the left.
5. The owner of the Green house drinks coffee.
6. The person who smokes Pall Mall rears birds.
7. The owner of the Yellow house smokes Dunhill.
8. The man living in the centre house drinks milk.
9. The Norwegian lives in the first house.
10. The man who smokes Blends lives next to the one who keeps cats.
11. The man who keeps horses lives next to the man who smokes Dunhill.
12. The man who smokes Blue

The puzzle is:



Master drinks beer.
13. The German smokes Prince.
14. The Norwegian lives next to the Blue house.
15. The man who smokes Blends has a neighbour who drinks water.

Variables

We label the houses with a number i between 1 and 5. The Boolean variable $S_{i,a}$ will be true exactly when the house with index i has attribute a.

```
Generate set of all variables
```

$\{S_{1,\mathit{Blends}}, S_{1,\mathit{Brit}}, S_{1,\mathit{Dane}}, S_{1,\mathit{Dunhill}}, S_{1,\mathit{German}}, S_{1,\mathit{Prince}}, S_{1,\mathit{Swede}}, S_{1,\mathit{beer}}, S_{1,\mathit{birds}}, S_{1,\mathit{blue}}, S_{1,\mathit{cats}},$
$S_{1,\mathit{coffee}}, S_{1,\mathit{dogs}}, S_{1,\mathit{fish}}, S_{1,\mathit{green}}, S_{1,\mathit{horses}}, S_{1,\mathit{milk}}, S_{1,\mathit{red}}, S_{1,\mathit{tea}}, S_{1,\mathit{water}}, S_{1,\mathit{white}}, S_{1,\mathit{yellow}},$
$S_{1,\mathit{BlueMaster}}, S_{1,\mathit{Norwegian}}, S_{1,\mathit{PallMall}}, S_{2,\mathit{Blends}}, S_{2,\mathit{Brit}}, S_{2,\mathit{Dane}}, S_{2,\mathit{Dunhill}}, S_{2,\mathit{German}}, S_{2,\mathit{Prince}},$
$S_{2,\mathit{Swede}}, S_{2,\mathit{beer}}, S_{2,\mathit{birds}}, S_{2,\mathit{blue}}, S_{2,\mathit{cats}}, S_{2,\mathit{coffee}}, S_{2,\mathit{dogs}}, S_{2,\mathit{fish}}, S_{2,\mathit{green}}, S_{2,\mathit{horses}}, S_{2,\mathit{milk}},$
$S_{2,\mathit{red}}, S_{2,\mathit{tea}}, S_{2,\mathit{water}}, S_{2,\mathit{white}}, S_{2,\mathit{yellow}}, S_{2,\mathit{BlueMaster}}, S_{2,\mathit{Norwegian}}, S_{2,\mathit{PallMall}}, S_{3,\mathit{Blends}}, S_{3,\mathit{Brit}},$
$S_{3,\mathit{Dane}}, S_{3,\mathit{Dunhill}}, S_{3,\mathit{German}}, S_{3,\mathit{Prince}}, S_{3,\mathit{Swede}}, S_{3,\mathit{beer}}, S_{3,\mathit{birds}}, S_{3,\mathit{blue}}, S_{3,\mathit{cats}}, S_{3,\mathit{coffee}}, S_{3,\mathit{dogs}},$
$S_{3,\mathit{fish}}, S_{3,\mathit{green}}, S_{3,\mathit{horses}}, S_{3,\mathit{milk}}, S_{3,\mathit{red}}, S_{3,\mathit{tea}}, S_{3,\mathit{water}}, S_{3,\mathit{white}}, S_{3,\mathit{yellow}}, S_{3,\mathit{BlueMaster}},$
$S_{3,\mathit{Norwegian}}, S_{3,\mathit{PallMall}}, S_{4,\mathit{Blends}}, S_{4,\mathit{Brit}}, S_{4,\mathit{Dane}}, S_{4,\mathit{Dunhill}}, S_{4,\mathit{German}}, S_{4,\mathit{Prince}}, S_{4,\mathit{Swede}},$
$S_{4,\mathit{beer}}, S_{4,\mathit{birds}}, S_{4,\mathit{blue}}, S_{4,\mathit{cats}}, S_{4,\mathit{coffee}}, S_{4,\mathit{dogs}}, S_{4,\mathit{fish}}, S_{4,\mathit{green}}, S_{4,\mathit{horses}}, S_{4,\mathit{milk}}, S_{4,\mathit{red}}, S_{4,\mathit{tea}},$
$S_{4,\mathit{water}}, S_{4,\mathit{white}}, S_{4,\mathit{yellow}}, S_{4,\mathit{BlueMaster}}, S_{4,\mathit{Norwegian}}, S_{4,\mathit{PallMall}}, S_{5,\mathit{Blends}}, S_{5,\mathit{Brit}}, S_{5,\mathit{Dane}},$
$S_{5,\mathit{Dunhill}}, S_{5,\mathit{German}}, S_{5,\mathit{Prince}}, S_{5,\mathit{Swede}}, S_{5,\mathit{beer}}, S_{5,\mathit{birds}}, S_{5,\mathit{blue}}, S_{5,\mathit{cats}}, S_{5,\mathit{coffee}}, S_{5,\mathit{dogs}}, S_{5,\mathit{fish}},$
$S_{5,\mathit{green}}, S_{5,\mathit{horses}}, S_{5,\mathit{milk}}, S_{5,\mathit{red}}, S_{5,\mathit{tea}}, S_{5,\mathit{water}}, S_{5,\mathit{white}}, S_{5,\mathit{yellow}}, S_{5,\mathit{BlueMaster}}, S_{5,\mathit{Norwegian}},$
$S_{5,\mathit{PallMall}}\}$

SAT encoding

Each attribute appears at least once

$$S_{1,\ blue} \lor S_{2,\ blue} \lor S_{3,\ blue} \lor S_{4,\ blue} \lor S_{5,\ blue}$$

No attribute appears twice

$$S_{1,\ blue} \Rightarrow \left(\neg S_{2,\ blue}\right)$$

For each attribute type every house has some attribute

$$S_{1,\ blue} \lor S_{1,\ green} \lor S_{1,\ red} \lor S_{1,\ white} \lor S_{1,\ yellow}$$

For each attribute type no house has multiple attributes

$$S_{1,\ blue} \Rightarrow \left(\neg S_{1,\ green}\right)$$

Encoding the known facts into logic

$$\begin{gathered}
S_{1,\ Brit} \Rightarrow S_{1,\ red}, S_{2,\ Brit} \Rightarrow S_{2,\ red}, S_{3,\ Brit} \Rightarrow S_{3,\ red}, S_{4,\ Brit} \Rightarrow S_{4,\ red}, S_{5,\ Brit} \Rightarrow S_{5,\ red}, S_{1,\ Swede} \\
\Rightarrow S_{1,\ dogs}, S_{2,\ Swede} \Rightarrow S_{2,\ dogs}, S_{3,\ Swede} \Rightarrow S_{3,\ dogs}, S_{4,\ Swede} \Rightarrow S_{4,\ dogs}, S_{5,\ Swede} \Rightarrow S_{5,\ dogs}, \\
S_{1,\ Dane} \Rightarrow S_{1,\ tea}, S_{2,\ Dane} \Rightarrow S_{2,\ tea}, S_{3,\ Dane} \Rightarrow S_{3,\ tea}, S_{4,\ Dane} \Rightarrow S_{4,\ tea}, S_{5,\ Dane} \Rightarrow S_{5,\ tea}, \\
S_{1,\ green} \Rightarrow S_{2,\ white}, S_{2,\ green} \Rightarrow S_{3,\ white}, S_{3,\ green} \Rightarrow S_{4,\ white}, S_{4,\ green} \Rightarrow S_{5,\ white}, \neg S_{5,\ green}, \\
S_{1,\ green} \Rightarrow S_{1,\ coffee}, S_{2,\ green} \Rightarrow S_{2,\ coffee}, S_{3,\ green} \Rightarrow S_{3,\ coffee}, S_{4,\ green} \Rightarrow S_{4,\ coffee}, S_{5,\ green} \\
\Rightarrow S_{5,\ coffee}, S_{1,\ PallMall} \Rightarrow S_{1,\ birds}, S_{2,\ PallMall} \Rightarrow S_{2,\ birds}, S_{3,\ PallMall} \Rightarrow S_{3,\ birds}, S_{4,\ PallMall} \\
\Rightarrow S_{4,\ birds}, S_{5,\ PallMall} \Rightarrow S_{5,\ birds}, S_{1,\ yellow} \Rightarrow S_{1,\ Dunhill}, S_{2,\ yellow} \Rightarrow S_{2,\ Dunhill}, S_{3,\ yellow} \\
\Rightarrow S_{3,\ Dunhill}, S_{4,\ yellow} \Rightarrow S_{4,\ Dunhill}, S_{5,\ yellow} \Rightarrow S_{5,\ Dunhill}, S_{3,\ milk}, S_{1,\ Norwegian}, S_{2,\ Blends} \\
\Rightarrow \left(S_{1,\ cats} \lor S_{3,\ cats}\right), S_{3,\ Blends} \Rightarrow \left(S_{2,\ cats} \lor S_{4,\ cats}\right), S_{4,\ Blends} \Rightarrow \left(S_{3,\ cats} \lor S_{5,\ cats}\right), \\
S_{1,\ Blends} \Rightarrow S_{2,\ cats}, S_{5,\ Blends} \Rightarrow S_{4,\ cats}, S_{2,\ horses} \Rightarrow \left(S_{1,\ Dunhill} \lor S_{3,\ Dunhill}\right), S_{3,\ horses} \\
\Rightarrow \left(S_{2,\ Dunhill} \lor S_{4,\ Dunhill}\right), S_{4,\ horses} \Rightarrow \left(S_{3,\ Dunhill} \lor S_{5,\ Dunhill}\right), S_{1,\ horses} \Rightarrow S_{2,\ Dunhill}, \\
S_{5,\ horses} \Rightarrow S_{4,\ Dunhill}, S_{1,\ BlueMaster} \Rightarrow S_{1,\ beer}, S_{2,\ BlueMaster} \Rightarrow S_{2,\ beer}, S_{3,\ BlueMaster} \Rightarrow S_{3,\ beer}, \\
S_{4,\ BlueMaster} \Rightarrow S_{4,\ beer}, S_{5,\ BlueMaster} \Rightarrow S_{5,\ beer}, S_{1,\ German} \Rightarrow S_{1,\ Prince}, S_{2,\ German} \Rightarrow S_{2,\ Prince}, \\
S_{3,\ German} \Rightarrow S_{3,\ Prince}, S_{4,\ German} \Rightarrow S_{4,\ Prince}, S_{5,\ German} \Rightarrow S_{5,\ Prince}, S_{2,\ Norwegian} \Rightarrow \left(S_{1,\ blue}\right. \\
\left.\lor S_{3,\ blue}\right), S_{3,\ Norwegian} \Rightarrow \left(S_{2,\ blue} \lor S_{4,\ blue}\right), S_{4,\ Norwegian} \Rightarrow \left(S_{3,\ blue} \lor S_{5,\ blue}\right),
\end{gathered}$$

$$S_{1,\,\mathit{Norwegian}} \Rightarrow S_{2,\,\mathit{blue}}, S_{5,\,\mathit{Norwegian}} \Rightarrow S_{4,\,\mathit{blue}}, S_{2,\,\mathit{Blends}} \Rightarrow \left(S_{1,\,\mathit{water}} \vee S_{3,\,\mathit{water}}\right), S_{3,\,\mathit{Blends}} \Rightarrow \left(S_{2,\,\mathit{water}} \vee S_{4,\,\mathit{water}}\right), S_{4,\,\mathit{Blends}} \Rightarrow \left(S_{3,\,\mathit{water}} \vee S_{5,\,\mathit{water}}\right), S_{1,\,\mathit{Blends}} \Rightarrow S_{2,\,\mathit{water}}, S_{5,\,\mathit{Blends}} \Rightarrow S_{4,\,\mathit{water}}$$

Finding a solution

We call **Satisfy** with all the constraints.

```
allConstraints := allAttributesAppear, noAttributesShared, allHaveEachAttribute, noneHaveMultiple, entries(c
satisfyingAssignment := Satisfy(&and(allConstraints));
```

$$\mathit{satisfyingAssignment} := \{S_{1,\,\mathit{Blends}} = \mathit{false}, S_{1,\,\mathit{Brit}} = \mathit{false}, S_{1,\,\mathit{Dane}} = \mathit{false}, S_{1,\,\mathit{Dunhill}} = \mathit{true}, S_{1,\,\mathit{German}} = \mathit{false}, S_{1,\,\mathit{Prince}} = \mathit{false}, S_{1,\,\mathit{Swede}} = \mathit{false}, S_{1,\,\mathit{beer}} = \mathit{false}, S_{1,\,\mathit{birds}} = \mathit{false}, S_{1,\,\mathit{blue}} = \mathit{false}, S_{1,\,\mathit{cats}} = \mathit{true}, S_{1,\,\mathit{coffee}} = \mathit{false}, S_{1,\,\mathit{dogs}} = \mathit{false}, S_{1,\,\mathit{fish}} = \mathit{false}, S_{1,\,\mathit{green}} = \mathit{false}, S_{1,\,\mathit{horses}} = \mathit{false}, S_{1,\,\mathit{milk}} = \mathit{false}, S_{1,\,\mathit{red}} = \mathit{false}, S_{1,\,\mathit{tea}} = \mathit{false}, S_{1,\,\mathit{water}} = \mathit{true}, S_{1,\,\mathit{white}} = \mathit{false}, S_{1,\,\mathit{yellow}} = \mathit{true}, S_{1,\,\mathit{BlueMaster}} = \mathit{false}, S_{1,\,\mathit{Norwegian}} = \mathit{true}, S_{1,\,\mathit{PallMall}} = \mathit{false}, S_{2,\,\mathit{Blends}} = \mathit{true}, S_{2,\,\mathit{Brit}} = \mathit{false}, S_{2,\,\mathit{Dane}} = \mathit{true}, S_{2,\,\mathit{Dunhill}} = \mathit{false}, S_{2,\,\mathit{German}} = \mathit{false}, S_{2,\,\mathit{Prince}} = \mathit{false}, S_{2,\,\mathit{Swede}} = \mathit{false}, S_{2,\,\mathit{beer}} = \mathit{false}, S_{2,\,\mathit{birds}} = \mathit{false}, S_{2,\,\mathit{blue}} = \mathit{true}, S_{2,\,\mathit{cats}} = \mathit{false}, S_{2,\,\mathit{coffee}} = \mathit{false}, S_{2,\,\mathit{dogs}} = \mathit{false}, S_{2,\,\mathit{fish}} = \mathit{false}, S_{2,\,\mathit{green}} = \mathit{false}, S_{2,\,\mathit{horses}} = \mathit{true}, S_{2,\,\mathit{milk}} = \mathit{false}, S_{2,\,\mathit{red}} = \mathit{false}, S_{2,\,\mathit{tea}} = \mathit{true}, S_{2,\,\mathit{water}} = \mathit{false}, S_{2,\,\mathit{white}} = \mathit{false}, S_{2,\,\mathit{yellow}} = \mathit{false}, S_{2,\,\mathit{BlueMaster}} = \mathit{false}, S_{2,\,\mathit{Norwegian}} = \mathit{false}, S_{2,\,\mathit{PallMall}} = \mathit{false}, S_{3,\,\mathit{Blends}} = \mathit{false}, S_{3,\,\mathit{Brit}} = \mathit{true}, S_{3,\,\mathit{Dane}} = \mathit{false}, S_{3,\,\mathit{Dunhill}} = \mathit{false}, S_{3,\,\mathit{German}} = \mathit{false}, S_{3,\,\mathit{Prince}} = \mathit{false}, S_{3,\,\mathit{Swede}} = \mathit{false}, S_{3,\,\mathit{beer}} = \mathit{false}, S_{3,\,\mathit{birds}} = \mathit{true}, S_{3,\,\mathit{blue}} = \mathit{false}, S_{3,\,\mathit{cats}} = \mathit{false}, S_{3,\,\mathit{coffee}} = \mathit{false}, S_{3,\,\mathit{dogs}} = \mathit{false}, S_{3,\,\mathit{fish}} = \mathit{false}, S_{3,\,\mathit{green}} = \mathit{false}, S_{3,\,\mathit{horses}} = \mathit{false}, S_{3,\,\mathit{milk}} = \mathit{true}, S_{3,\,\mathit{red}} = \mathit{true}, S_{3,\,\mathit{tea}} = \mathit{false}, S_{3,\,\mathit{water}} = \mathit{false}, S_{3,\,\mathit{white}} = \mathit{false}, S_{3,\,\mathit{yellow}} = \mathit{false}, S_{3,\,\mathit{BlueMaster}} = \mathit{false}, S_{3,\,\mathit{Norwegian}} = \mathit{false}, S_{3,\,\mathit{PallMall}} = \mathit{true}, S_{4,\,\mathit{Blends}} = \mathit{false}, S_{4,\,\mathit{Brit}} = \mathit{false}, S_{4,\,\mathit{Dane}} = \mathit{false}, S_{4,\,\mathit{Dunhill}} = \mathit{false}, S_{4,\,\mathit{German}} = \mathit{true}, S_{4,\,\mathit{Prince}} = \mathit{true}, S_{4,\,\mathit{Swede}} = \mathit{false}, S_{4,\,\mathit{beer}} = \mathit{false}, S_{4,\,\mathit{birds}} = \mathit{false}, S_{4,\,\mathit{blue}} = \mathit{false}, S_{4,\,\mathit{cats}} = \mathit{false}, S_{4,\,\mathit{coffee}} = \mathit{true}, S_{4,\,\mathit{dogs}} = \mathit{false}, S_{4,\,\mathit{fish}} = \mathit{true}, S_{4,\,\mathit{green}} = \mathit{true}, S_{4,\,\mathit{horses}} = \mathit{false}, S_{4,\,\mathit{milk}} = \mathit{false}, S_{4,\,\mathit{red}} = \mathit{false}, S_{4,\,\mathit{tea}} = \mathit{false}, S_{4,\,\mathit{water}} = \mathit{false}, S_{4,\,\mathit{white}} = \mathit{false}, S_{4,\,\mathit{yellow}} = \mathit{false}, S_{4,\,\mathit{BlueMaster}} = \mathit{false}, S_{4,\,\mathit{Norwegian}} = \mathit{false}, S_{4,\,\mathit{PallMall}} = \mathit{false}, S_{5,\,\mathit{Blends}} = \mathit{false}, S_{5,\,\mathit{Brit}} = \mathit{false}, S_{5,\,\mathit{Dane}} = \mathit{false}, S_{5,\,\mathit{Dunhill}} = \mathit{false}, S_{5,\,\mathit{German}} = \mathit{false}, S_{5,\,\mathit{Prince}} = \mathit{false}, S_{5,\,\mathit{Swede}} = \mathit{true}, S_{5,\,\mathit{beer}} = \mathit{true}, S_{5,\,\mathit{birds}} = \mathit{false}, S_{5,\,\mathit{blue}} = \mathit{false}, S_{5,\,\mathit{cats}} = \mathit{false}, S_{5,\,\mathit{coffee}} = \mathit{false}, S_{5,\,\mathit{dogs}} = \mathit{true}, S_{5,\,\mathit{fish}} = \mathit{false}, S_{5,\,\mathit{green}} = \mathit{false}, S_{5,\,\mathit{horses}} = \mathit{false}, S_{5,\,\mathit{milk}} = \mathit{false}, S_{5,\,\mathit{red}} = \mathit{false}, S_{5,\,\mathit{tea}} = \mathit{false}, S_{5,\,\mathit{water}} = \mathit{false}, S_{5,\,\mathit{white}} = \mathit{true}, S_{5,\,\mathit{yellow}} = \mathit{false}, S_{5,\,\mathit{BlueMaster}} = \mathit{true}, S_{5,\,\mathit{Norwegian}} = \mathit{false}, S_{5,\,\mathit{PallMall}} = \mathit{false}\}$$

To easily see who lives where and the attributes of each house and owner we fill a 2D array with data from the found solution:

```
Read satisfying assignment and produce nicely formatted solu
```

$$\begin{bmatrix} \textit{yellow} & \textit{blue} & \textit{red} & \textit{green} & \textit{white} \\ \textit{Norwegian} & \textit{Dane} & \textit{Brit} & \textit{German} & \textit{Swede} \\ \textit{water} & \textit{tea} & \textit{milk} & \textit{coffee} & \textit{beer} \\ \textit{Dunhill} & \textit{Blends} & \textit{PallMall} & \textit{Prince} & \textit{BlueMaster} \\ \textit{cats} & \textit{horses} & \textit{birds} & \textit{fish} & \textit{dogs} \end{bmatrix}$$

Euler's Graeco-Latin Square Conjecture

A *Latin square* of order n is an $n \times n$ matrix containing n distinct entries such that every row and column does not contain duplicate entries.

$$\begin{bmatrix} \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} \\ \mathrm{B} & \mathrm{A} & \mathrm{D} & \mathrm{C} \\ \mathrm{C} & \mathrm{D} & \mathrm{A} & \mathrm{B} \\ \mathrm{D} & \mathrm{C} & \mathrm{B} & \mathrm{A} \end{bmatrix}$$

A superposition of two Latin squares is called a *Graeco-Latin square* if all n^2 pairs of entries appear.

$$\begin{bmatrix} A\alpha & B\gamma & C\delta & D\beta \\ B\beta & A\delta & D\gamma & C\alpha \\ C\gamma & D\alpha & A\beta & B\delta \\ D\delta & C\beta & B\alpha & A\gamma \end{bmatrix}$$

History

In 1782 Euler constructed Graeco-Latin squares in all orders n except those of the form $4k+2$. He conjectured that no Graeco-Latin squares exist in these orders.

In 1900, Tarry proved that Graeco-Latin squares do not exist in order six. Three independent invalid proofs of Euler's conjecture were published in 1902, 1910, and 1922.

However, in 1959–1960 Bose, Shrikhande, and Parker showed these proofs were flawed by constructing Graeco-Latin squares in all orders except 2 and 6.

Variables

The Boolean variables $X_{i,j,k}$ will represent that the (i,j)th entry of the first Latin square is k (and the variables $Y_{i,j,k}$ will be used for the second Latin square).

```
Generate set of variables
```

$$\{X_{1,1,A}, X_{1,1,B}, X_{1,1,C}, X_{1,1,\mathrm{D}}, X_{1,2,A}, X_{1,2,B}, X_{1,2,C}, X_{1,2,\mathrm{D}}, X_{1,3,A}, X_{1,3,B}, X_{1,3,C}, X_{1,3,\mathrm{D}},$$
$$X_{1,4,A}, X_{1,4,B}, X_{1,4,C}, X_{1,4,\mathrm{D}}, X_{2,1,A}, X_{2,1,B}, X_{2,1,C}, X_{2,1,\mathrm{D}}, X_{2,2,A}, X_{2,2,B}, X_{2,2,C}, X_{2,2,\mathrm{D}},$$
$$X_{2,3,A}, X_{2,3,B}, X_{2,3,C}, X_{2,3,\mathrm{D}}, X_{2,4,A}, X_{2,4,B}, X_{2,4,C}, X_{2,4,\mathrm{D}}, X_{3,1,A}, X_{3,1,B}, X_{3,1,C}, X_{3,1,\mathrm{D}},$$
$$X_{3,2,A}, X_{3,2,B}, X_{3,2,C}, X_{3,2,\mathrm{D}}, X_{3,3,A}, X_{3,3,B}, X_{3,3,C}, X_{3,3,\mathrm{D}}, X_{3,4,A}, X_{3,4,B}, X_{3,4,C}, X_{3,4,\mathrm{D}},$$
$$X_{4,1,A}, X_{4,1,B}, X_{4,1,C}, X_{4,1,\mathrm{D}}, X_{4,2,A}, X_{4,2,B}, X_{4,2,C}, X_{4,2,\mathrm{D}}, X_{4,3,A}, X_{4,3,B}, X_{4,3,C}, X_{4,3,\mathrm{D}},$$
$$X_{4,4,A}, X_{4,4,B}, X_{4,4,C}, X_{4,4,\mathrm{D}}, Y_{1,1,A}, Y_{1,1,B}, Y_{1,1,C}, Y_{1,1,\mathrm{D}}, Y_{1,2,A}, Y_{1,2,B}, Y_{1,2,C}, Y_{1,2,\mathrm{D}},$$
$$Y_{1,3,A}, Y_{1,3,B}, Y_{1,3,C}, Y_{1,3,\mathrm{D}}, Y_{1,4,A}, Y_{1,4,B}, Y_{1,4,C}, Y_{1,4,\mathrm{D}}, Y_{2,1,A}, Y_{2,1,B}, Y_{2,1,C}, Y_{2,1,\mathrm{D}},$$
$$Y_{2,2,A}, Y_{2,2,B}, Y_{2,2,C}, Y_{2,2,\mathrm{D}}, Y_{2,3,A}, Y_{2,3,B}, Y_{2,3,C}, Y_{2,3,\mathrm{D}}, Y_{2,4,A}, Y_{2,4,B}, Y_{2,4,C}, Y_{2,4,\mathrm{D}},$$
$$Y_{3,1,A}, Y_{3,1,B}, Y_{3,1,C}, Y_{3,1,\mathrm{D}}, Y_{3,2,A}, Y_{3,2,B}, Y_{3,2,C}, Y_{3,2,\mathrm{D}}, Y_{3,3,A}, Y_{3,3,B}, Y_{3,3,C}, Y_{3,3,\mathrm{D}},$$
$$Y_{3,4,A}, Y_{3,4,B}, Y_{3,4,C}, Y_{3,4,\mathrm{D}}, Y_{4,1,A}, Y_{4,1,B}, Y_{4,1,C}, Y_{4,1,\mathrm{D}}, Y_{4,2,A}, Y_{4,2,B}, Y_{4,2,C}, Y_{4,2,\mathrm{D}},$$
$$Y_{4,3,A}, Y_{4,3,B}, Y_{4,3,C}, Y_{4,3,\mathrm{D}}, Y_{4,4,A}, Y_{4,4,B}, Y_{4,4,C}, Y_{4,4,\mathrm{D}}\}$$

SAT encoding

```
Each location contains at least one entry
```

$$X_{1,1,A} \vee X_{1,1,B} \vee X_{1,1,C} \vee X_{1,1,\mathrm{D}}$$

```
Each location contains at most one entry
```

$$X_{1,1,A} \Rightarrow (\neg X_{1,1,B})$$

```
All columns and rows contain distinct entries
```

$$X_{1,1,A} \Rightarrow (\neg X_{1,2,A})$$

```
Each pair of entries (e.g., A and alpha) must appear somewhe
```

$$(X_{1,1,A} \wedge Y_{1,1,\alpha}) \vee (X_{1,2,A} \wedge Y_{1,2,\alpha}) \vee (X_{1,3,A} \wedge Y_{1,3,\alpha}) \vee (X_{1,4,A} \wedge Y_{1,4,\alpha}) \vee (X_{2,1,A} \wedge Y_{2,1,\alpha}) \vee (X_{2,2,A} \wedge Y_{2,2,\alpha}) \vee (X_{2,3,A} \wedge Y_{2,3,\alpha}) \vee (X_{2,4,A} \wedge Y_{2,4,\alpha}) \vee (X_{3,1,A} \wedge Y_{3,1,\alpha}) \vee (X_{3,2,A} \wedge Y_{3,2,\alpha}) \vee (X_{3,3,A} \wedge Y_{3,3,\alpha}) \vee (X_{3,4,A} \wedge Y_{3,4,\alpha}) \vee (X_{4,1,A} \wedge Y_{4,1,\alpha}) \vee (X_{4,2,A} \wedge Y_{4,2,\alpha}) \vee (X_{4,3,A} \wedge Y_{4,3,\alpha}) \vee (X_{4,4,A} \wedge Y_{4,4,\alpha})$$

```
Can assume certain entries of the Latin squares by permuting
```

$$X_{1,1,A} \wedge X_{1,2,B} \wedge X_{1,3,C} \wedge X_{1,4,\mathrm{D}} \wedge X_{2,1,B} \wedge X_{3,1,C} \wedge X_{4,1,\mathrm{D}} \wedge Y_{1,1,\alpha} \wedge Y_{1,2,\beta} \wedge Y_{1,3,\gamma} \wedge Y_{1,4,\delta}$$

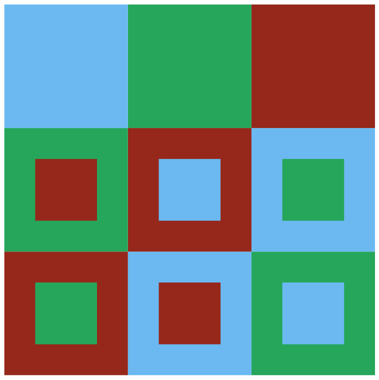
Finding Graeco-Latin squares

We use **Satisfy** to find a satisfying assignment of all the constraints, **RealTime** from the **CodeTools** package to time how long the computation takes and commands from the **plots** and **plottools** packages to display the solution.

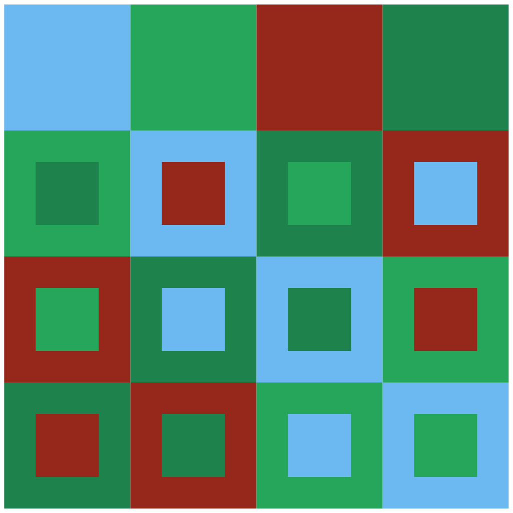
```
Commands to plot Graeco-Latin squares and find them for smal
```

```
Found a Graeco-Latin square of order 1 in 0.00 seconds:
```

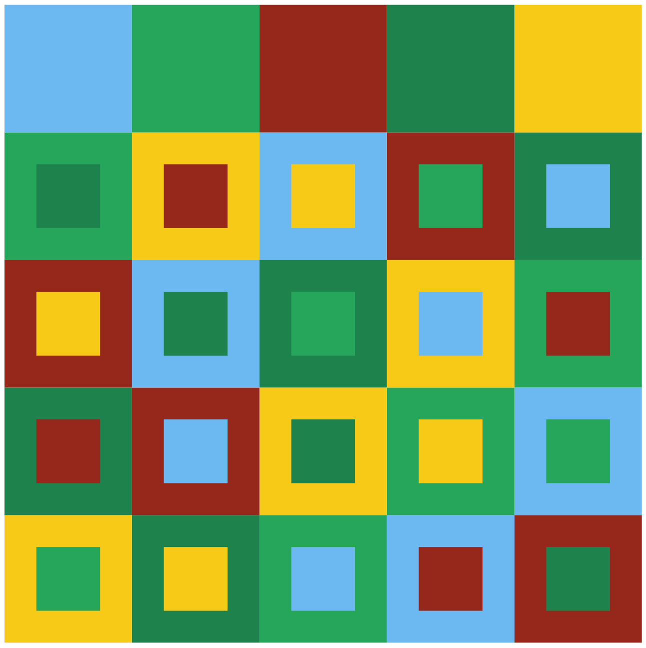
```
Found that no Graeco-Latin squares of order 2 exist in 0.01
seconds.
Found a Graeco-Latin square of order 3 in 0.02 seconds:
```



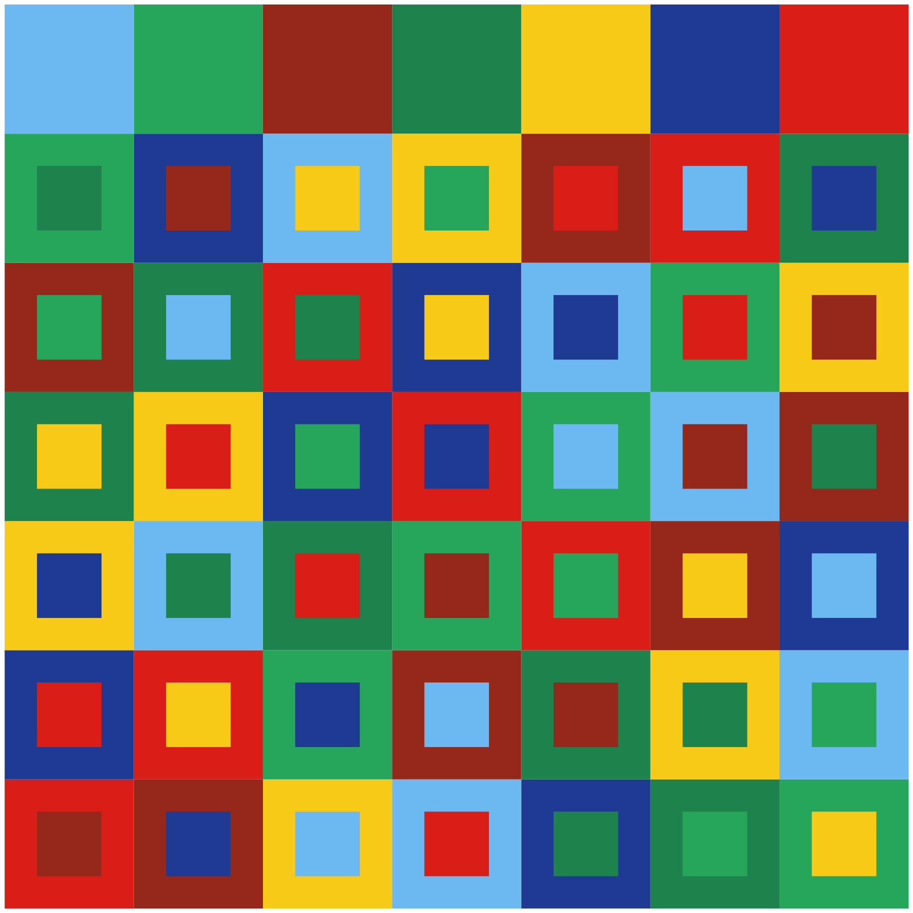
Found a Graeco-Latin square of order 4 in 0.15 seconds:



Found a Graeco-Latin square of order 5 in 0.13 seconds:

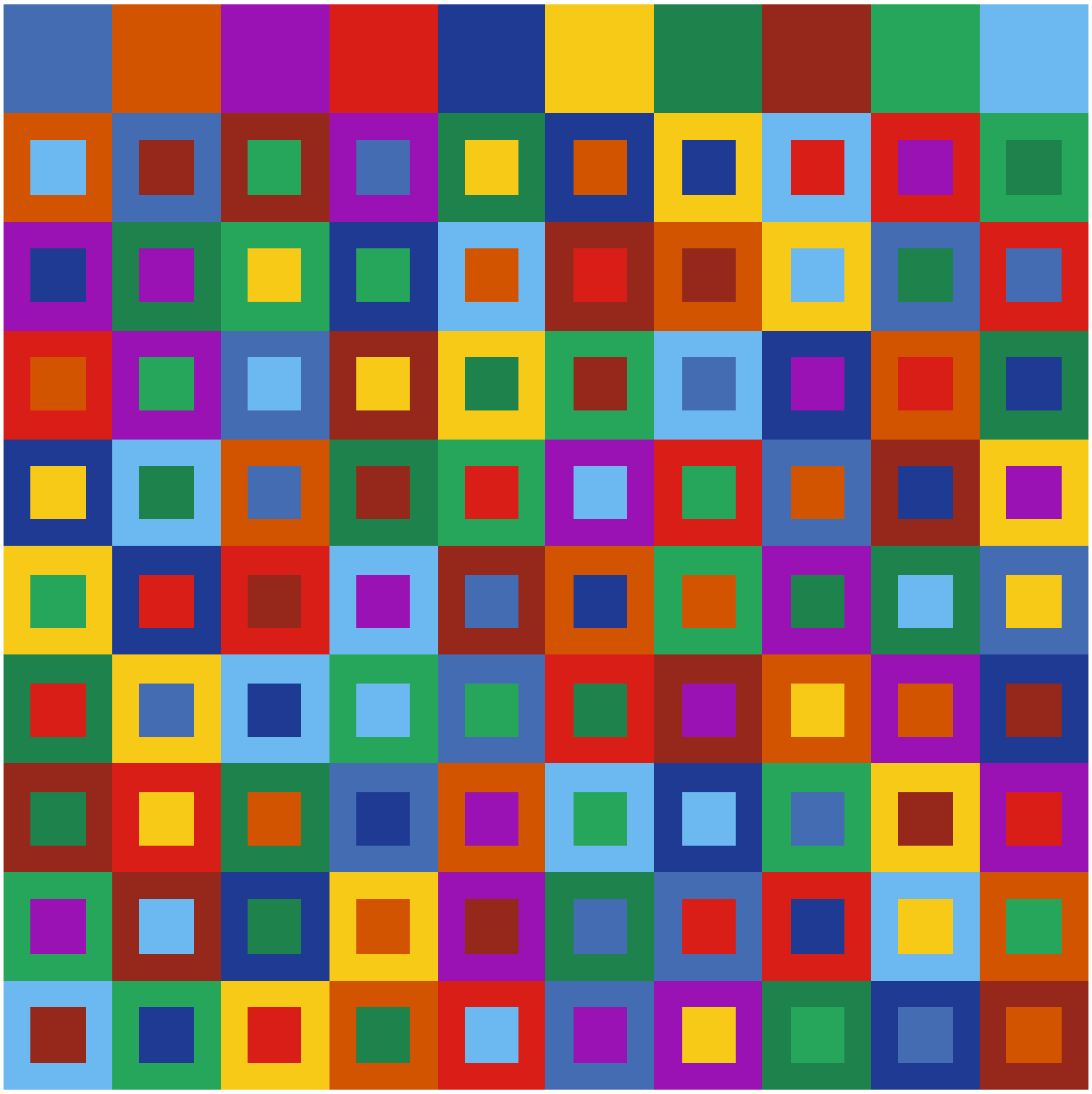


Found a Graeco-Latin square of order 7 in 0.72 seconds:



Counterexample to Euler's conjecture

Using this encoding Maple finds a Graeco-Latin square of order ten in about a day:



Generating Sudoku puzzles

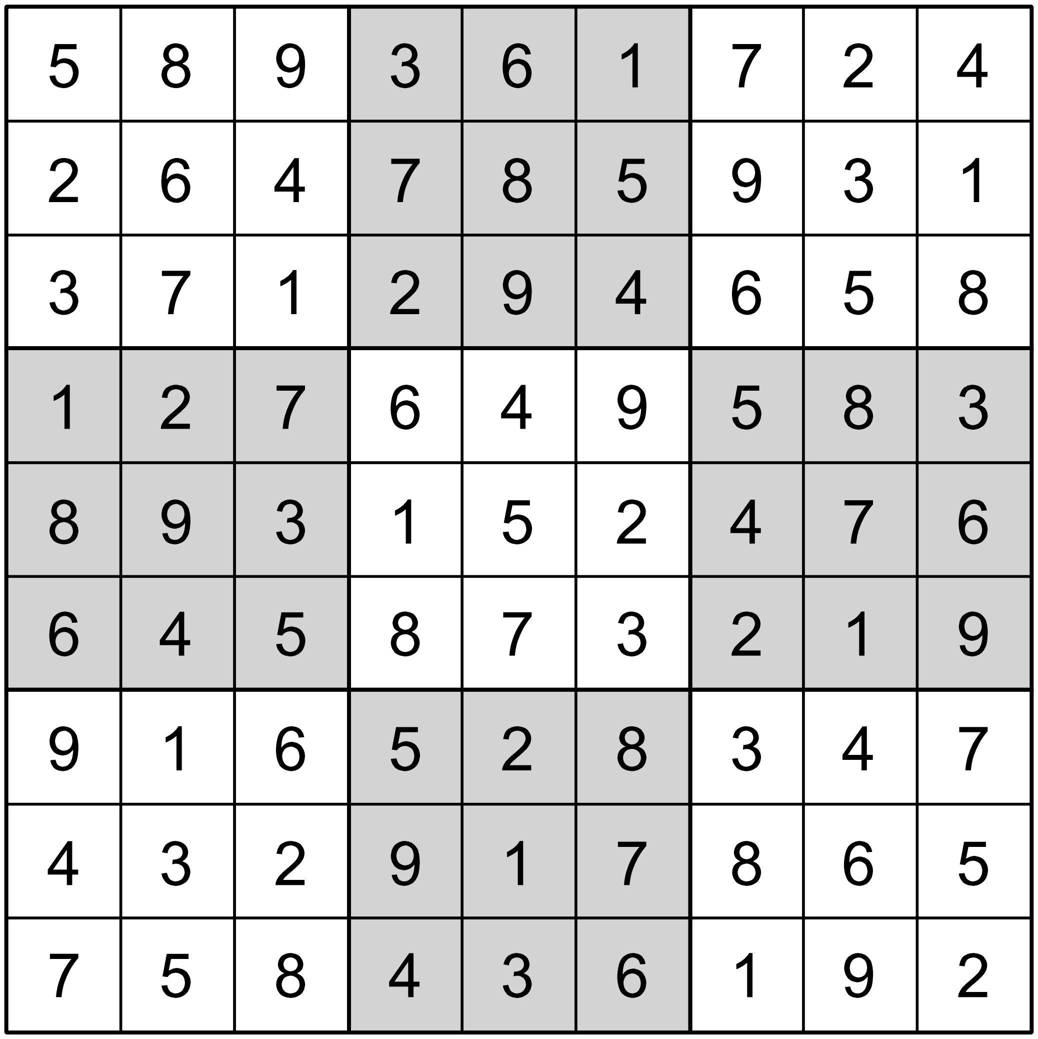
8								
		3	6					
	7			9		2		
	5				7			
				4	5	7		
			1				3	
		1					6	8
		8	5				1	
	9					4		

A similar encoding can be used to generate a random Sudoku puzzles. A completed Sudoku grid is a 9 by 9 Latin square such that the entries in each highlighted 3 by 3 block are distinct as well.

First, we call **Satisfy** with the Sudoku constraints (with no initial entries). A random assignment is generated by passing a random seed to the SAT solver using **solveroptions** in the **Satisfy** call.

```
constraints := &and(atLeastOneDigit, distinctnessConstraints):
satisfyingAssignment := Satisfy(constraints, solveroptions=[rnd_init_act=true, random_seed=floor(1000*time[r

numberOfEntriesToThrowAway := 0:
```

Commands to plot a Sudoku puzzle

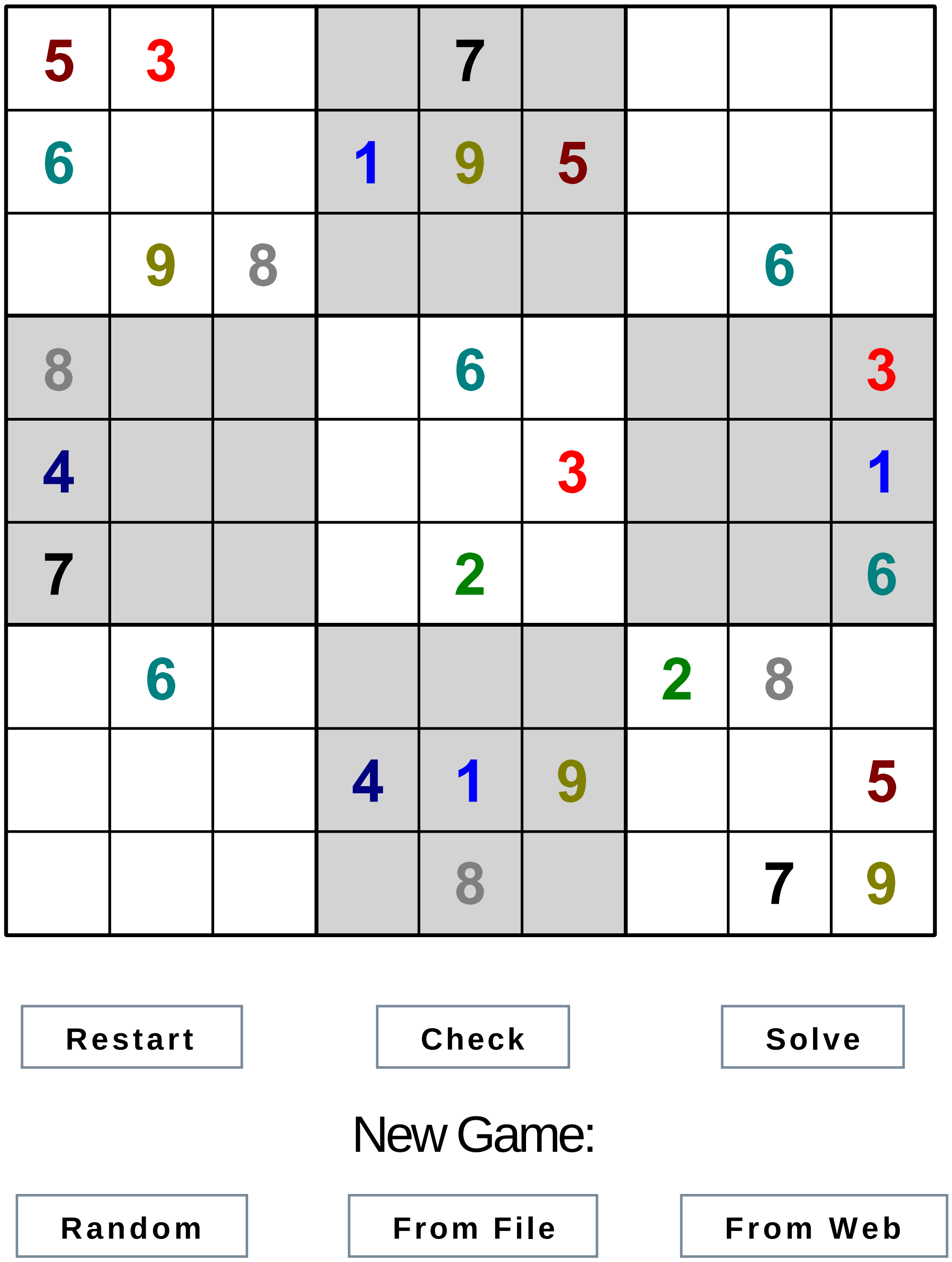


5	8	9	3	6	1	7	2	4
2	6	4	7	8	5	9	3	1
3	7	1	2	9	4	6	5	8
1	2	7	6	4	9	5	8	3
8	9	3	1	5	2	4	7	6
6	4	5	8	7	3	2	1	9
9	1	6	5	2	8	3	4	7
4	3	2	9	1	7	8	6	5
7	5	8	4	3	6	1	9	2

This is a valid Sudoku puzzle.

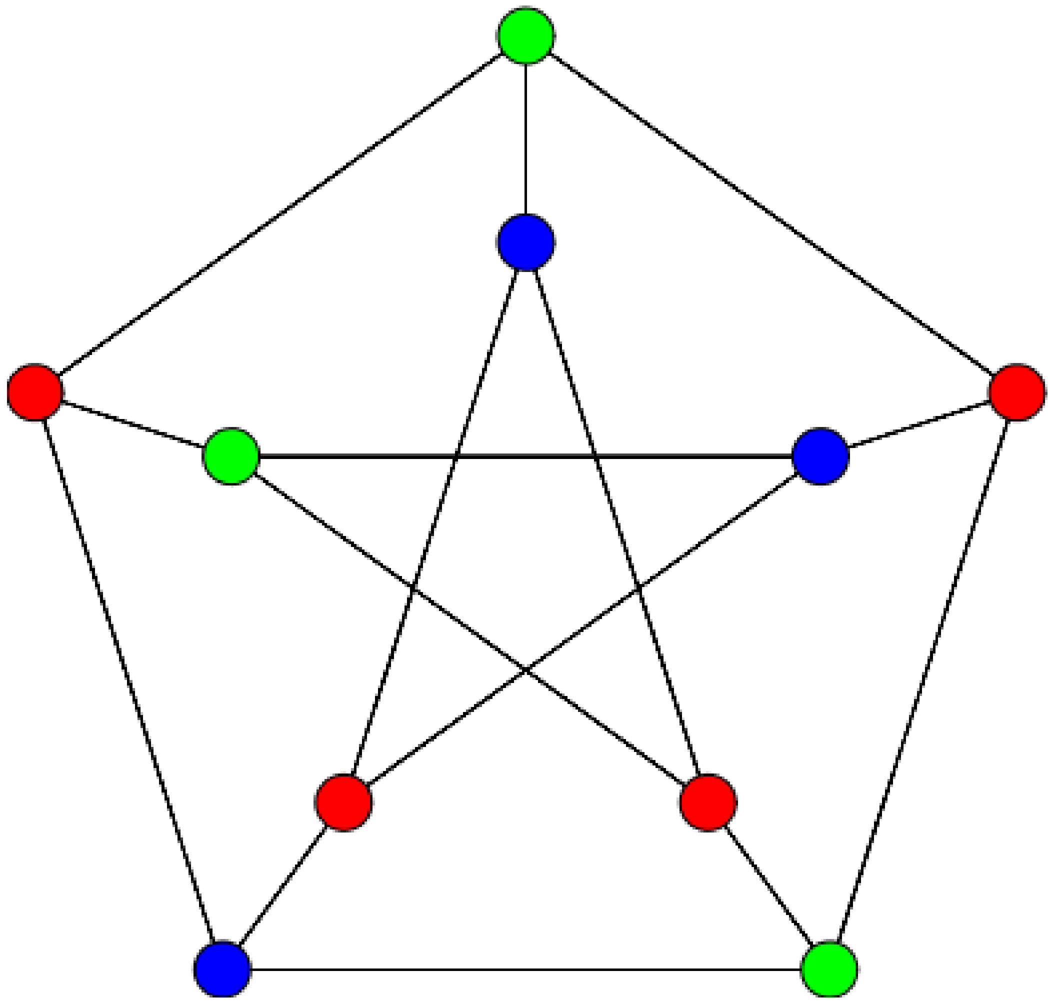
Using this method of generating Sudoku puzzles we implemented an interactive Sudoku game.

Interactive Sudoku by Curtis Bright



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4					3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

The Graph Colouring Problem



A *colouring* of a graph is an assignment of colours to its vertices such that every two adjacent vertices are coloured differently.

A minimal colouring of a graph can be computed using the **ChromaticNumber** function of the **GraphTheory** package.

Variables

Let

$x_{i,j}$ represent that vertex j of a graph G is coloured with colour i.

SAT encoding

Each vertex has been assigned a colour

$$x_{1,\,red} \vee x_{1,\,blue} \vee x_{1,\,green}$$

No vertex has been assigned two colours

$$(\neg x_{1,\,red}) \vee (\neg x_{1,\,blue})$$

Adjacent vertices are not coloured the same way

$$(\neg x_{1,\,red}) \vee (\neg x_{2,\,red})$$

Finding a colouring

We call **Satisfy** on the constraints with k starting at 1 and increase k until the set of constraints becomes satisfiable.

Solving a benchmark using a random graph

G := Graph 1: an undirected unweighted graph with 125 vertices and 736 edge(s)

```
Could not find a colouring of G with 1 colours.  Total time: 0.40
seconds.
Could not find a colouring of G with 2 colours.  Total time: 1.08
seconds.
Could not find a colouring of G with 3 colours.  Total time: 2.09
seconds.
Could not find a colouring of G with 4 colours.  Total time: 3.31
seconds.
Found a colouring of G with 5 colours.  Total time: 4.89 seconds.

The graph G was coloured using 5 colours in 4.89 seconds.
```

ChromaticNumber function

As of Maple 2018, a tuned version of this approach has been implemented in the **ChromaticNumber** function.

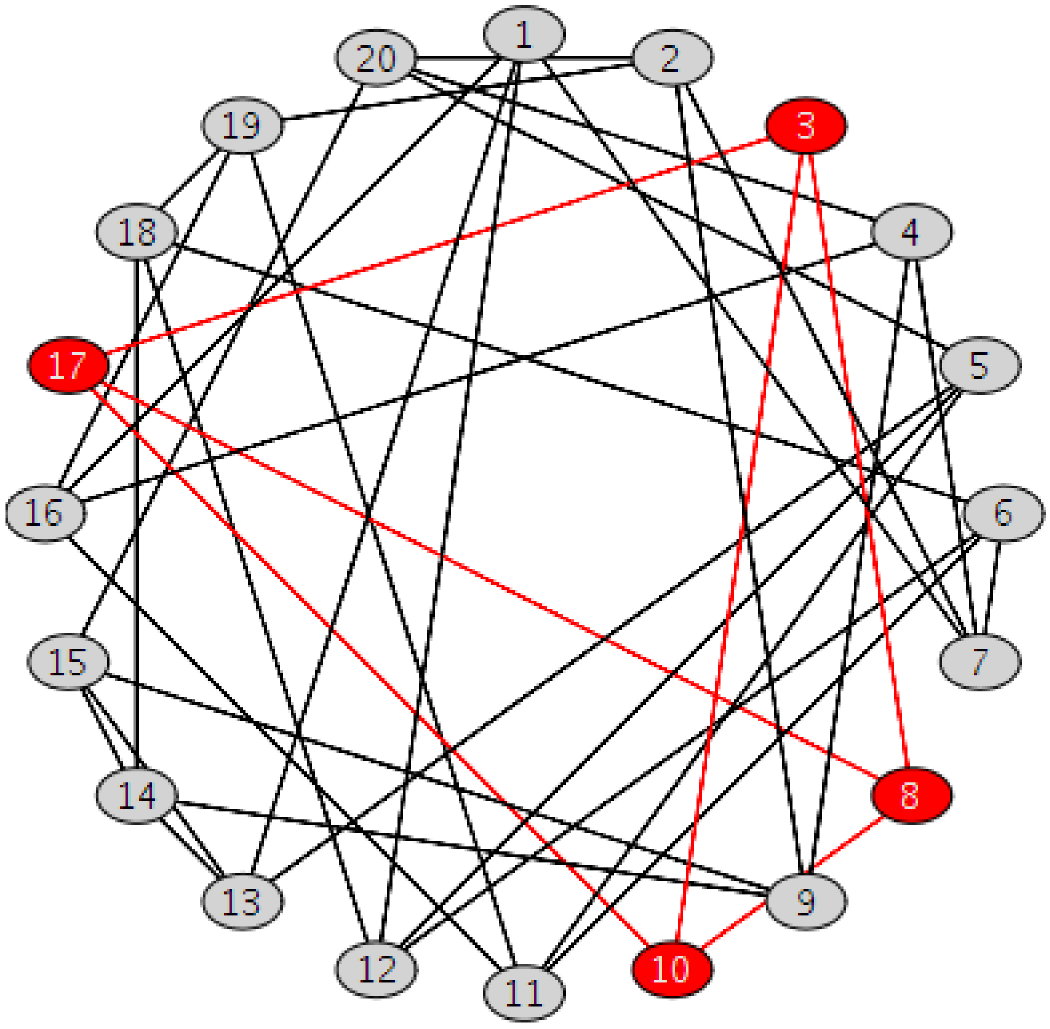
```
G := :-Import("example/DSJC125.1.s6", base = datadir);
computationTime, chi := CodeTools:-RealTime(ChromaticNumber(G, method=sat)):

printf("Maple 2019 finds the chromatic number of G to be %d in %.2f seconds.", chi, computationTime);
```

$$G := \textit{Graph 2: an undirected unweighted graph with 125 vertices and 736 edge(s)}$$

```
Maple 2019 finds the chromatic number of G to be 5 in 0.11
seconds.
```

Prior to 2018, Maple could not find the chromatic number of G after an hour!

The Maximum Clique Problem



A *clique* of a graph is a subset of its vertices that are all mutually connected.

The *maximum clique problem* is to find a clique of maximum size in a given graph and can be solved with the **MaximumClique** function of the **GraphTheory** package.

Variables

We use the Boolean variables x_i where i is a vertex of G to represent if the vertex i appears in the clique we are trying to find (say we are trying to find a clique of size k).

SAT encoding

If the vertices 1 and 2 are *not* connected then the variables x_1 and x_2 cannot both be true.

```
Generate clique connectedness constraints
```

$$(\neg x_1) \vee (\neg x_2)$$

We also want a way of enforcing that at least k variables $x_1, \ldots, x_n$ are assigned to true.

The most basic way of doing this is something like $\left(x_1 \wedge \cdots \wedge x_k\right) \vee \cdots \vee \left(x_{n-k+1} \wedge \cdots \wedge x_n\right)$, but this is very inefficient.

Clever size encoding

Instead, we use Boolean counter variables $s_{k,\,n}$ that represent at least k of the variables $x_1, \ldots, x_n$ are assigned true. This occurs exactly when:

1. Either at least k of $x_1, \ldots, x_{n-1}$ are true, or
2. at least $k-1$ of $x_1, \ldots, x_{n-1}$ are true and x_n is true.

This gives an efficient recursive definition of $s_{k,\,n}$ in terms of $s_{k,\,n-1}$ and $s_{k-1,\,n-1}$.

```
Boolean counter constraints with k=3 and n=5
```

$$s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}, s_{4,0}, s_{5,0}, \neg s_{0,1}, \neg s_{0,2}, \neg s_{0,3}, s_{1,1} \Leftrightarrow \left(s_{0,1} \vee \left(x_1 \wedge s_{0,0}\right)\right), s_{2,1} \Leftrightarrow \left(s_{1,1} \vee \left(x_2 \wedge s_{1,0}\right)\right), s_{3,1} \Leftrightarrow \left(s_{2,1} \vee \left(x_3 \wedge s_{2,0}\right)\right), s_{4,1} \Leftrightarrow \left(s_{3,1} \vee \left(x_4 \wedge s_{3,0}\right)\right), s_{5,1} \Leftrightarrow \left(s_{4,1} \vee \left(x_5 \wedge s_{4,0}\right)\right), s_{1,2} \Leftrightarrow \left(s_{0,2} \vee \left(x_1 \wedge s_{0,1}\right)\right), s_{2,2} \Leftrightarrow \left(s_{1,2} \vee \left(x_2 \wedge s_{1,1}\right)\right), s_{3,2} \Leftrightarrow \left(s_{2,2} \vee \left(x_3 \wedge s_{2,1}\right)\right), s_{4,2} \Leftrightarrow \left(s_{3,2} \vee \left(x_4 \wedge s_{3,1}\right)\right), s_{5,2} \Leftrightarrow \left(s_{4,2} \vee \left(x_5 \wedge s_{4,1}\right)\right), s_{1,3} \Leftrightarrow \left(s_{0,3} \vee \left(x_1 \wedge s_{0,2}\right)\right), s_{2,3} \Leftrightarrow \left(s_{1,3} \vee \left(x_2 \wedge s_{1,2}\right)\right), s_{3,3} \Leftrightarrow \left(s_{2,3} \vee \left(x_3 \wedge s_{2,2}\right)\right), s_{4,3} \Leftrightarrow \left(s_{3,3} \vee \left(x_4 \wedge s_{3,2}\right)\right), s_{5,3} \Leftrightarrow \left(s_{4,3} \vee \left(x_5 \wedge s_{4,2}\right)\right), s_{5,3}$$

Finding a maximum clique

We call **Satisfy** on the constraints with k starting at 1 and increase k until the set of constraints becomes unsatisfiable.

```
Solving a benchmark from the DIMACS implementation challenge
```

$G :=$ *Graph 3: an undirected unweighted graph with 45 vertices and 918 edge(s)*

```
Found a clique of G of size 1.  Total time: 0.02 seconds.
Found a clique of G of size 2.  Total time: 0.05 seconds.
Found a clique of G of size 3.  Total time: 0.09 seconds.
Found a clique of G of size 4.  Total time: 0.14 seconds.
Found a clique of G of size 5.  Total time: 0.20 seconds.
Found a clique of G of size 6.  Total time: 0.27 seconds.
Found a clique of G of size 7.  Total time: 0.43 seconds.
Found a clique of G of size 8.  Total time: 0.52 seconds.
Found a clique of G of size 9.  Total time: 0.63 seconds.
Found a clique of G of size 10.  Total time: 0.75 seconds.
Found a clique of G of size 11.  Total time: 0.95 seconds.
Found a clique of G of size 12.  Total time: 1.09 seconds.
Found a clique of G of size 13.  Total time: 1.25 seconds.
Found a clique of G of size 14.  Total time: 1.49 seconds.
Found a clique of G of size 15.  Total time: 1.67 seconds.
Found a clique of G of size 16.  Total time: 1.94 seconds.
Could not find a clique of G of size 17.  Total time: 2.23
seconds.

The largest clique of G has size 16 and it was found in 2.23
seconds.
```

MaximumClique function

As of Maple 2019, a tuned version of this approach has been implemented in the **MaximumClique** function.

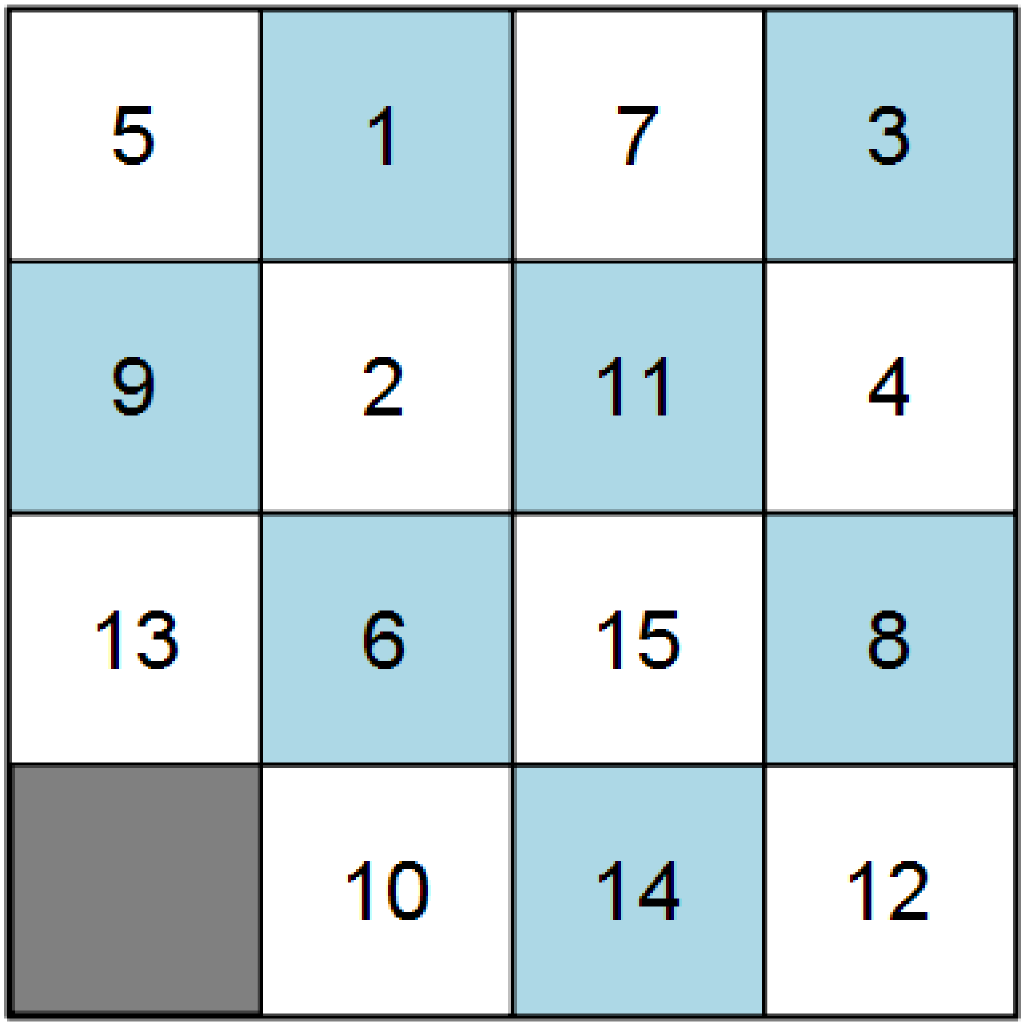
```
G := :-Import("example/MANN_a9.clq", base=datadir, format="dimacs");
computationTime, clique := CodeTools:-RealTime(MaximumClique(G, method=sat)):
printf("The largest clique of G has size %d and it was found in %.2f seconds.\n", nops(clique), computationT
```

$$G := \textit{Graph 4: an undirected unweighted graph with 45 vertices and 918 edge(s)}$$

```
The largest clique of G has size 16 and it was found in 0.07
seconds.
```

Prior to 2019, Maple required about 225 seconds to find a maximum clique in this graph.

The 15-puzzle



The 15-puzzle is a classic "sliding tile" puzzle. It consists of a 4×4 grid containing tiles numbered 1 through 15 along with one missing tile.

The objective is to order the tiles in increasing order (from left to right and top to bottom) and end with the blank tile in the lower-right.

Variables

We'll use the Boolean variables $S[i, j, n, t]$ to denote that the square (i, j) contains tile n after t moves.

SAT encoding

```
Constraints directly specifying state of above puzzle
```

$$S_{1,1,5,0} \wedge S_{2,1,9,0} \wedge S_{3,1,13,0} \wedge S_{4,1,blank,0} \wedge S_{1,2,1,0} \wedge S_{2,2,2,0} \wedge S_{3,2,6,0} \wedge S_{4,2,10,0} \wedge S_{1,3,7,0} \wedge S_{2,3,11,0} \wedge S_{3,3,15,0} \wedge S_{4,3,14,0} \wedge S_{1,4,3,0} \wedge S_{2,4,4,0} \wedge S_{3,4,8,0} \wedge S_{4,4,12,0}$$

```
Constraints directly specifying that the board is solved at
```

$$S_{1,1,1,t} \wedge S_{2,1,5,t} \wedge S_{3,1,9,t} \wedge S_{4,1,13,t} \wedge S_{1,2,2,t} \wedge S_{2,2,6,t} \wedge S_{3,2,10,t} \wedge S_{4,2,14,t} \wedge S_{1,3,3,t} \wedge S_{2,3,7,t} \wedge S_{3,3,11,t} \wedge S_{4,3,15,t} \wedge S_{1,4,4,t} \wedge S_{2,4,8,t} \wedge S_{3,4,12,t} \wedge S_{4,4,blank,t}$$

```
Two tiles cannot occupy the same square at the same time
```

$$(\neg S_{1,1,1,0}) \vee (\neg S_{1,1,2,0})$$

The rules of the puzzle imply that the tile in square (i, j) does not change when (i, j) and its neighbours are not blank, i.e.,

$$notEqualOrAdjacentToBlank(i, j, t) \Rightarrow doesNotChange(i, j, t)$$

for all timesteps t.

```
Generate the "static" transition constraints
```

$$((\neg S_{1,1,blank,0}) \wedge (\neg S_{1,2,blank,0}) \wedge (\neg S_{2,1,blank,0})) \Rightarrow ((S_{1,1,1,0} \Leftrightarrow S_{1,1,1,1}) \wedge (S_{1,1,2,0} \Leftrightarrow S_{1,1,2,1}) \wedge (S_{1,1,3,0} \Leftrightarrow S_{1,1,3,1}) \wedge (S_{1,1,4,0} \Leftrightarrow S_{1,1,4,1}) \wedge (S_{1,1,5,0} \Leftrightarrow S_{1,1,5,1}) \wedge (S_{1,1,6,0} \Leftrightarrow S_{1,1,6,1}) \wedge (S_{1,1,7,0} \Leftrightarrow S_{1,1,7,1}) \wedge (S_{1,1,8,0} \Leftrightarrow S_{1,1,8,1}) \wedge (S_{1,1,9,0} \Leftrightarrow S_{1,1,9,1}) \wedge (S_{1,1,10,0} \Leftrightarrow S_{1,1,10,1}) \wedge (S_{1,1,11,0} \Leftrightarrow S_{1,1,11,1}) \wedge (S_{1,1,12,0} \Leftrightarrow S_{1,1,12,1}) \wedge (S_{1,1,13,0} \Leftrightarrow S_{1,1,13,1}) \wedge (S_{1,1,14,0} \Leftrightarrow S_{1,1,14,1}) \wedge (S_{1,1,15,0}$$

$$\Leftrightarrow S_{1,1,15,1}) \wedge (S_{1,1,blank,0} \Leftrightarrow S_{1,1,blank,1}))$$

If $oneTileMoved(i, j, k, l, t)$ represents that only square (i, j) moves to (k, l) at timestep t then we also know the constraint

$$S[i, j, blank, t] \Rightarrow \bigvee_{(k,l)} oneTileMoved(i, j, k, l, t)$$

where (k, l) is adjacent to (i, j).

```
Generate the "slide" transition constraints
```

$$S_{1,1,blank,0} \Rightarrow (((S_{1,2,1,0} \Leftrightarrow S_{1,1,1,1}) \wedge (S_{1,2,2,0} \Leftrightarrow S_{1,1,2,1}) \wedge (S_{1,2,3,0} \Leftrightarrow S_{1,1,3,1}) \wedge (S_{1,2,4,0} \Leftrightarrow S_{1,1,4,1}) \wedge (S_{1,2,5,0} \Leftrightarrow S_{1,1,5,1}) \wedge (S_{1,2,6,0} \Leftrightarrow S_{1,1,6,1}) \wedge (S_{1,2,7,0} \Leftrightarrow S_{1,1,7,1}) \wedge (S_{1,2,8,0} \Leftrightarrow S_{1,1,8,1}) \wedge (S_{1,2,9,0} \Leftrightarrow S_{1,1,9,1}) \wedge (S_{1,2,10,0} \Leftrightarrow S_{1,1,10,1}) \wedge (S_{1,2,11,0} \Leftrightarrow S_{1,1,11,1}) \wedge (S_{1,2,12,0} \Leftrightarrow S_{1,1,12,1}) \wedge (S_{1,2,13,0} \Leftrightarrow S_{1,1,13,1}) \wedge (S_{1,2,14,0} \Leftrightarrow S_{1,1,14,1}) \wedge (S_{1,2,15,0} \Leftrightarrow S_{1,1,15,1}) \wedge (S_{1,2,blank,0} \Leftrightarrow S_{1,1,blank,1})) \wedge ((S_{2,1,1,0} \Leftrightarrow S_{2,1,1,1}) \wedge (S_{2,1,2,0} \Leftrightarrow S_{2,1,2,1}) \wedge (S_{2,1,3,0} \Leftrightarrow S_{2,1,3,1}) \wedge (S_{2,1,4,0} \Leftrightarrow S_{2,1,4,1}) \wedge (S_{2,1,5,0} \Leftrightarrow S_{2,1,5,1}) \wedge (S_{2,1,6,0} \Leftrightarrow S_{2,1,6,1}) \wedge (S_{2,1,7,0} \Leftrightarrow S_{2,1,7,1}) \wedge (S_{2,1,8,0} \Leftrightarrow S_{2,1,8,1}) \wedge (S_{2,1,9,0} \Leftrightarrow S_{2,1,9,1}) \wedge (S_{2,1,10,0} \Leftrightarrow S_{2,1,10,1}) \wedge (S_{2,1,11,0} \Leftrightarrow S_{2,1,11,1}) \wedge (S_{2,1,12,0} \Leftrightarrow S_{2,1,12,1}) \wedge (S_{2,1,13,0} \Leftrightarrow S_{2,1,13,1}) \wedge (S_{2,1,14,0} \Leftrightarrow S_{2,1,14,1}) \wedge (S_{2,1,15,0} \Leftrightarrow S_{2,1,15,1}) \wedge (S_{2,1,blank,0} \Leftrightarrow S_{2,1,blank,1})))$$

Finding a solution

We ask the SAT solver to find a solution up to M (starting at 5) moves. If no solution is found, increase M.

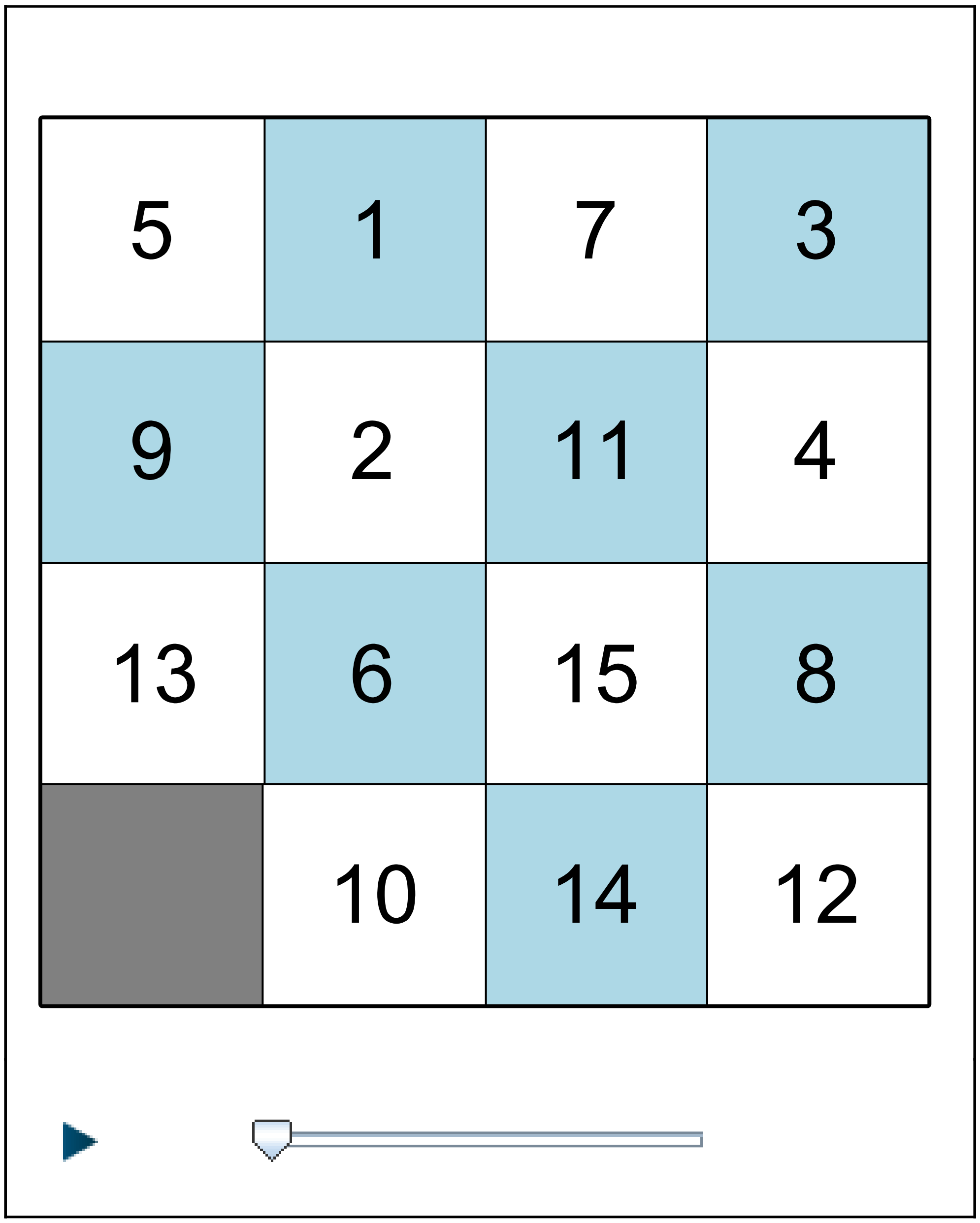
```
Commands to search for a solution
```

```
Generated 11697 constraints with 1536 variables and now searching
for a solution with at most 5 moves...
 No solution found with at most 5 moves in 1.44 seconds.
Generated 21457 constraints with 2816 variables and now searching
```

```
for a solution with at most 10 moves...
No solution found with at most 10 moves in 2.86 seconds.
Generated 31217 constraints with 4096 variables and now searching
for a solution with at most 15 moves...
Solution found with at most 15 moves in 6.51 seconds.
```

We'll use the **Explore** command to allow us to see how the board state changes over time with a slider controlling what time t to view.

```
Commands for reading and plotting the solution with the Expl
```



Conclusion

SAT solvers can be surprisingly useful for some problems!

Our SAT encodings for **ChromaticNumber** and **MaximumClique** often outperform Maple's previous own built-in functions (sometimes significantly so).

By default, Maple 2019 will run the SAT approach and the traditional Maple approach on separate cores and return the result of whichever finishes first.

It's worthwhile to add the **Satisfy** command into your toolbox of useful Maple functions.