Constraint Satisfaction Programming

The PEQNP System

Author: Oscar Riveros

Institute: www.PEQNP.science

Date: February 4, 2020

"NULLIUS IN VERBA"

Contents

0.1 The 8-Queens Problem i
0.2 Send More Money ii
0.3 Magic Square iii

0.1 The 8-Queens Problem

The N-Queens problem originates from a question relating to chess, The 8-Queens problem. Chess is played on an 8×8 grid, with each piece taking up one cell. A queen is a piece in chess that, in any given move, can move any distance vertically, horizontally, or diagonally. However, the queen cannot move more than one direction per turn. It can only move one direction per turn. So a question one might ask themselves is whether or not you can place 8 queens on a chessboard so that none of the queens can kill each other in one move (i.e.There is no way for a queen to in one cell to reach a queen in another cell in one move)?

```
>>> # Import all PEQNP System.
>>> from peqnp import *
>>> # The number of Queens.
>>> n = 8
>>> # Initialize the engine with the max number represented
>>> # on the problem and if include the - sign add 1.
>>> engine(bits=n.bit_length() + 1)
>>> # A n size vector of integers.
>>> qs = vector(size=n)
>>> # All number on the solution are different.
>>> all_different(qs)
>>> # All numbers on the solution are < n.
>>> apply_single(qs, lambda x: x < n)
>>> # All the qs[i] + 1 and qs[i] - i are different by pairs.
>>> apply_dual([qs[i] + i for i in range(n)], lambda x, y: x != y)
>>> apply_dual([qs[i] - i for i in range(n)], lambda x, y: x != y)
>>> # Is satisfiable?.
>>> satisfy()
True
>>> # Do something with the solution.
>>> for i in range(n):
...     print(''.join(['Q ' if qs[i] == j else '. ' for j in range(n)]))
...
. Q . . . . . .
. . . . . . Q .
. . . . Q . . .
. . . . . . . Q
Q . . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .
```

0.2 Send More Money

The Send More Money Problem consists in finding distinct digits for the letters D, E, M, N, O, R, S, Y such that S and M are different from zero (no leading zeros) and the equation

$$SEND + MORE = MONEY$$

is satisfied.

```
>>> # Import all PEQNP System.
>>> from peqnp import *
>>> # Initialize the engine with the max number represented.
>>> engine(32)
>>> # A set of variables.
>>> abc = [s, e, n, d, m, o, r, y] = vector(size=8)
>>> # All variables are different.
>>> all_different(abc)
>>> # All variables are in {0..9}
>>> apply_single(abc, lambda x: x <= 9)
>>> # The constraints.
>>> e1 = 1000 * s + 100 * e + 10 * n + d
>>> e2 = 1000 * m + 100 * o + 10 * r + e
>>> e3 = 10000 * m + 1000 * o + 100 * n + 10 * d + y
>>> assert e1 + e2 == e3
>>> assert s != 0
>>> assert m != 0
>>> # Print all solutions.
>>> while satisfy():
...     print(abc)
...
[9, 4, 5, 3, 1, 0, 8, 7]
[9, 4, 5, 2, 1, 0, 7, 6]
[9, 3, 4, 2, 1, 0, 8, 5]
[9, 5, 6, 3, 1, 0, 7, 8]
[9, 6, 7, 2, 1, 0, 5, 8]
[9, 7, 8, 5, 1, 0, 6, 2]
```

0.3 Magic Square

A magic square is an arrangement of distinct integers in an $N \times N$ grid, where the numbers in each row, and in each column, and the numbers in the main and secondary diagonals, all add up to the same number.

```
>>> # Interaction with other 3rd libraries, NUMPY in this case
>>> import numpy as np
>>> # Import all PEQNP System
>>> from peqnp import *
>>> # A 3x3 magic square.
>>> n = 3
>>> # With 10 bit engine is sufficient.
>>> engine(10)
>>> # The magic constant.
>>> c = integer()
>>> # The magic square.
>>> X = np.asarray(matrix(dimensions=(n, n)))
>>> # All elements are different.
>>> all_different(X.flatten())
>>> # All elements are in {1..9}
>>> apply_single(X.flatten(), lambda x: 0 < x < 10)
>>> # The main constraints.
>>> for row, col in zip(X, X.T):
...     # Rows
...     assert sum(row) == c
...     # Columns
...     assert sum(col) == c
...
>>> # Diagonals
>>> assert sum(X[i][i] for i in range(n)) == c
>>> assert sum(X[i][n - 1 - i] for i in range(n)) == c

>>> # Satisfy with turbo, this force to SALIME X
>>> # to do a full simplification destroying the problem,
>>> # so only ine solution can get with turbo.
>>> if satisfy():
...     print(X)
...
[[8 3 4]
 [1 5 9]
 [6 7 2]]
```