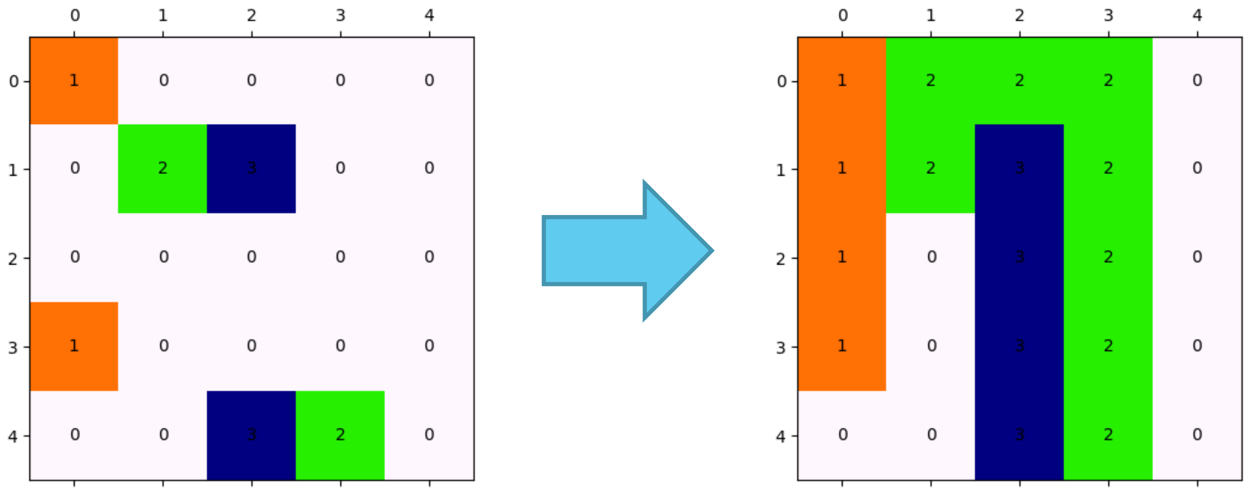
Routing with SAT

By: Roger Pujol Torramorell

The problem

- 2D Grid
- Pairs of dots
- Connect all pairs without creating any short circuit among them



SAT (Pseudo Boolean)

- **Python3.7** for the encoding of the problem to Pseudo Boolean
- **PBlib** to encode the Pseudo Boolean into actual SAT
- **Minisat** as the SAT solver
- **Python3** to post-process the results

PB: Variables

- One 2D grid of Boolean variables for each pair of dots
- Each grid indicates the dots and their connection for a particular pair
- Basically 3D grid with axes X, Y and P where P is the id of a pair of dots.

PB: Constraints (informal)

- Force the dots from each pair to be 1
- Force that all the positions in the 2D grid (X,Y) can only have one pair level set to 1
- Force that the initial dots have exactly one neighbour set to 1
- Force that all positions without initial dots:
  - If 1 then have exactly two neighbours set to 1
  - Else 0

PB: Constraints

- For each dot *D* in pair:
  - $D = 1$
  - $C_{left\_of\_D} + C_{above\_of\_D} + C_{right\_of\_D} + C_{under\_of\_D} = 1$
- For each position in the X,Y axis being *C* cell and *k* the last pair id:
  - $C_1 + C_2 + \ldots + C_k \leq 1$
- For each position *C* in the 3D grid (excluding the initial dots)
  - $C_{left\_of\_C} + C_{above\_of\_C} + C_{right\_of\_C} + C_{under\_of\_C} + 3\neg C \geq 2$
  - $C_{left\_of\_C} + C_{above\_of\_C} + C_{right\_of\_C} + C_{under\_of\_C} + 3\neg C \leq 2$

Problems

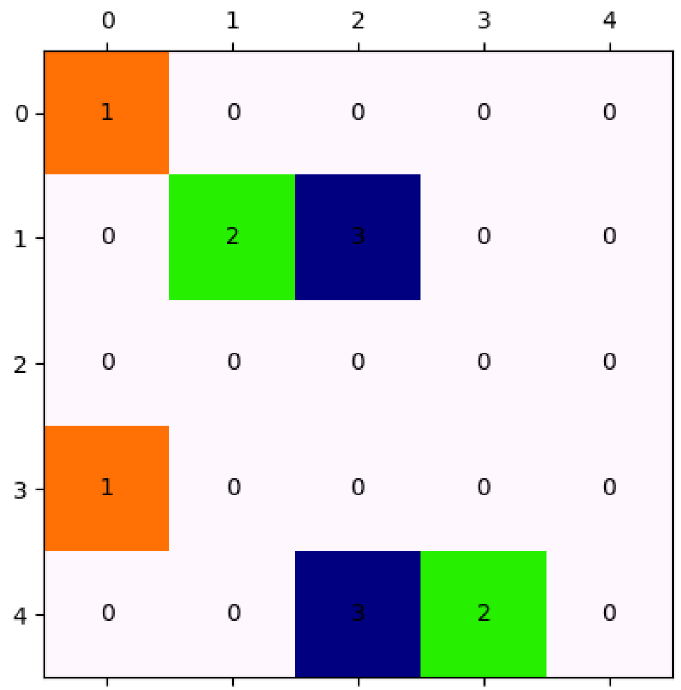
- This gives a not optimum solution.
- This encoding ensures connections between pairs of dots but also there might be cycles without any dot.
  - Can be solved in a simple post processing

- Both problems can be solved by optimizing

Optimization

- Limit the number of positions in the 3D grid that are set to 1
- Constraint to limit:
  - $\sum_{C \in 3DGrid} C \leq limit$
- Binary search to find the lowest limit possible
  - Minimum is equal to the sum Manhattan Distances of all pairs
  - Maximum is equal to the whole 2D Grid

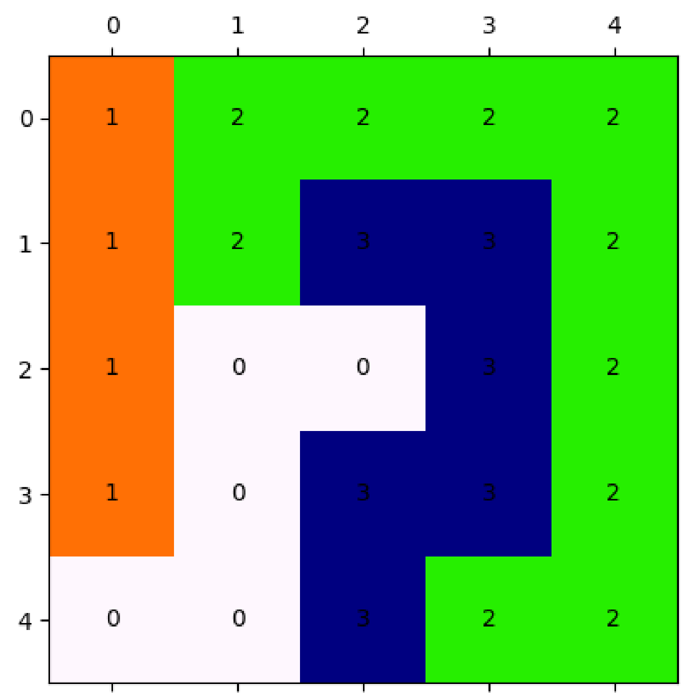
Example



Example

```
* #variable= 75 #constraint= 151
1 x1 >= 1;
1 x4 >= 1;
+1 x2 +1 x6  = 1;
+1 x3 +1 x5 +1 x9  = 1;
…
-1 x1 -1 x26 -1 x51 >= -1;
…
-1 x26 -1 x32 -1 x36  +3 ~x31 >= -2;
+1 x26 +1 x32 +1 x36  +3 ~x31 >= 2;
…
```

Example

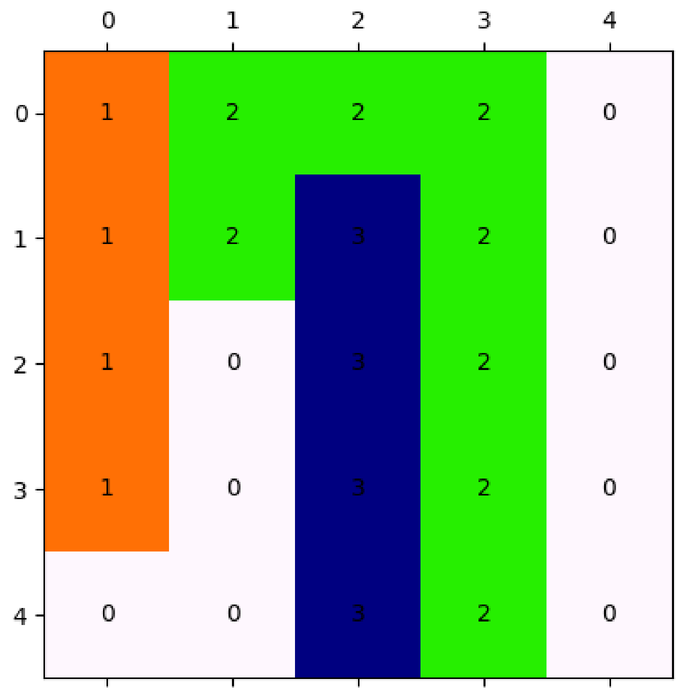


Example Optimization

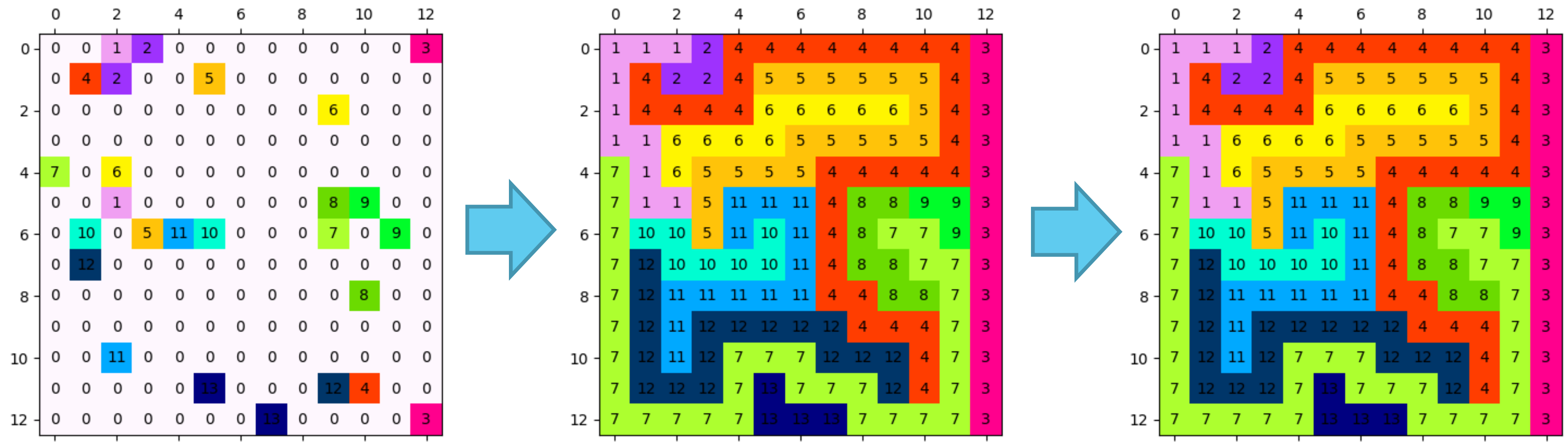
- Cost=25…SAT
- Cost=14…UNSAT
- Cost=19…SAT
- Cost=16...SAT
- Cost=15...UNSAT
- Best SAT solution Cost=16

- Constraint added to limit (limit=16):
  - -1 x1 -1 x2 -1 x3 … -1 x73 -1 x74 -1 x75>= -16;

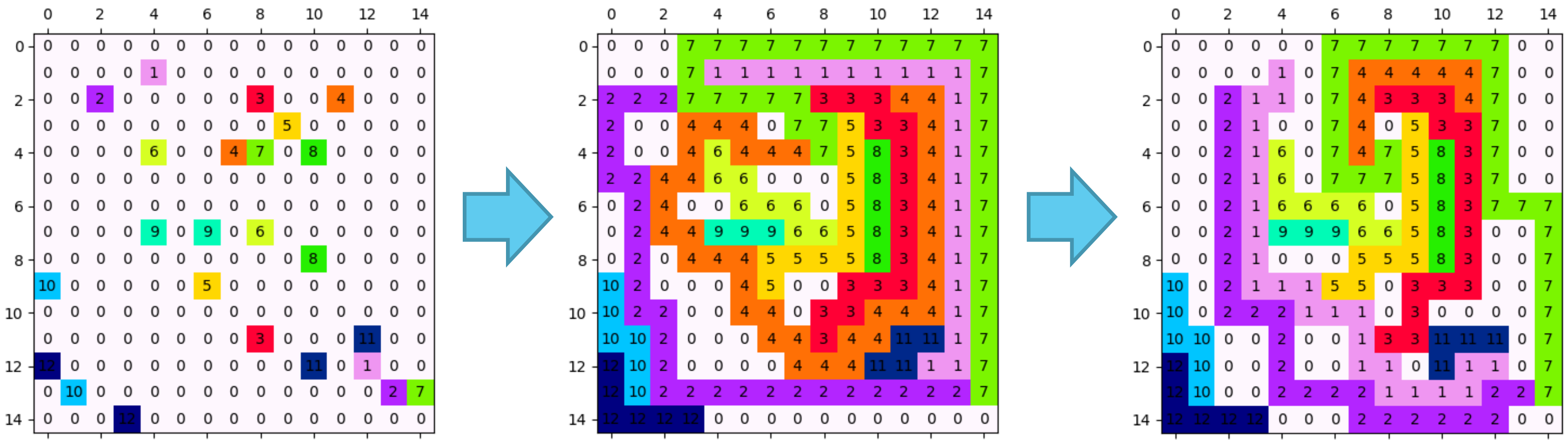
Example Optimization



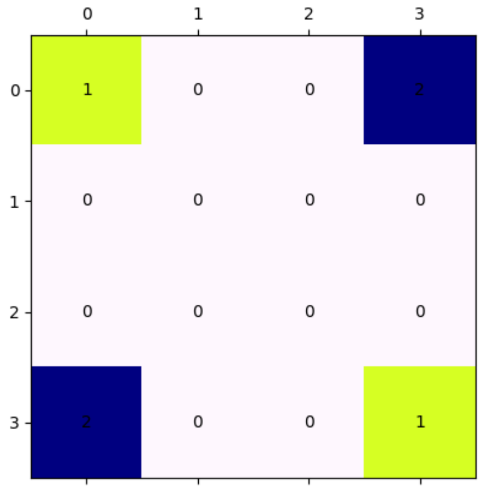
Other Examples (1/3)

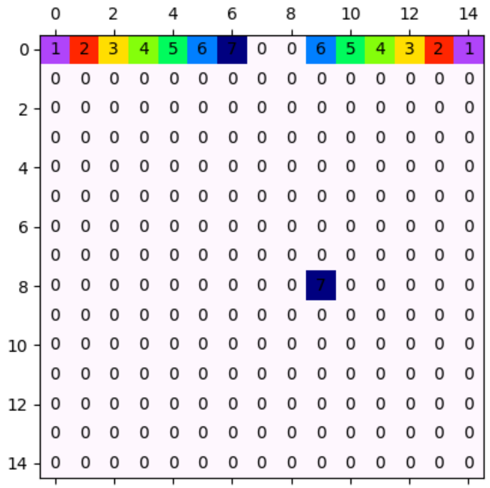


Other Examples (2/3)



Other Examples (3/3)





Demo

- Give me an input
- We run the program
- We check the output