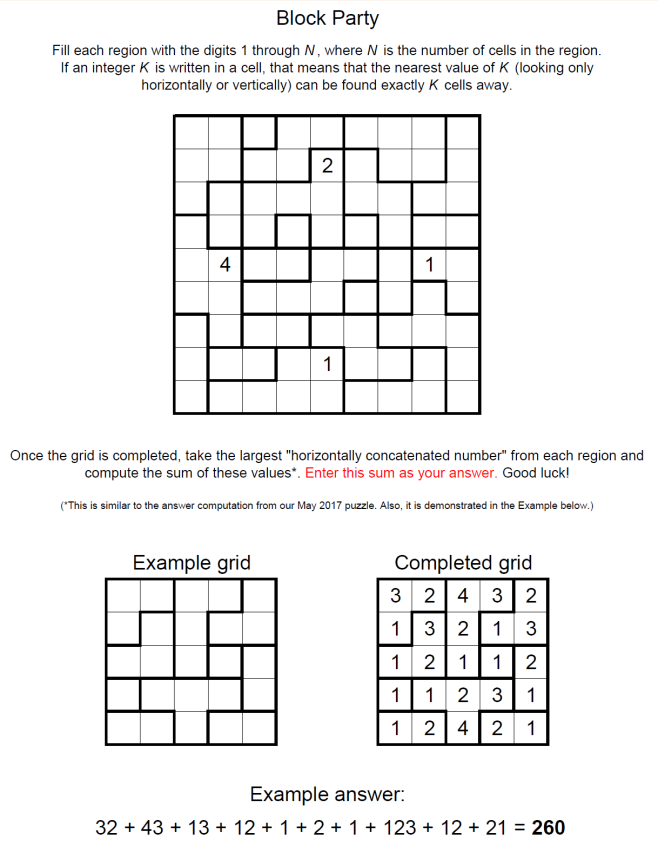
Vamshi Jandhyala

Block Party

✠

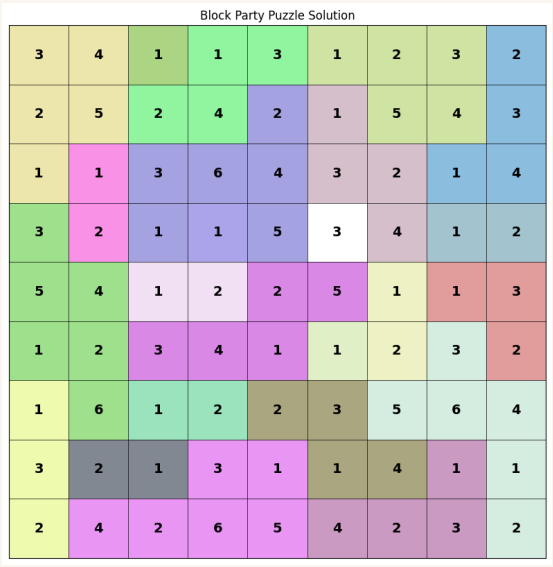
2024



A Jane Street puzzle. Fill each region with the numbers 1 through N, where N is the number of cells in the region. For every number K in the grid, the nearest K via taxicab distance must be exactly K cells away.

Solution

Using the Python code below, which makes use of the Z3 library, we obtain the solution shown.



Python code

```
from z3 import *
import random
from matplotlib import pyplot as plt

def solve_block_party(grid):
    solver = Solver()
    n = 9
    cells = [[Int(f"cell_{i}_{j}") for j in range(n)] for i in range(n)]

    for i in range(n):
        for j in range(n):
            solver.add(And(1 <= cells[i][j], cells[i][j] <= 9))

    for item in grid:
        if isinstance(item, tuple):
            i, j, value = item
            solver.add(cells[i][j] == value)
        elif isinstance(item, list):
            size = len(item)
            solver.add(Distinct([cells[x][y] for x, y in item]))
            for x, y in item:
                solver.add(And(1 <= cells[x][y], cells[x][y] <= size))

    for i in range(n):
        for j in range(n):
            rule_constraints = []
            for k in range(1, 10):
                if j >= k:
                    rule_constraints.append(And(
                        cells[i][j] == k, cells[i][j - k] == k,
                        And([cells[i][jj] != k for jj in range(j - k + 1, j)])))
                if j + k < n:
                    rule_constraints.append(And(
                        cells[i][j] == k, cells[i][j + k] == k,
                        And([cells[i][jj] != k for jj in range(j + 1, j + k)])))
```

```
                if i >= k:
                    rule_constraints.append(And(
                        cells[i][j] == k, cells[i - k][j] == k,
                        And([cells[ii][j] != k for ii in range(i - k + 1, i)])))
                if i + k < n:
                    rule_constraints.append(And(
                        cells[i][j] == k, cells[i + k][j] == k,
                        And([cells[ii][j] != k for ii in range(i + 1, i + k)])))
            solver.add(Or(rule_constraints))

    if solver.check() == sat:
        model = solver.model()
        return [[model.evaluate(cells[i][j]).as_long() for j in range(n)]
                for i in range(n)]
    return None

grid = [
    (1, 4, 2), (4, 1, 4), (4, 7, 1), (7, 4, 1),
    [[0,0],[0,1],[1,0],[1,1],[2,0]],
    [[2,1],[3,1]], [[0,2]], [[0,3],[0,4],[1,2],[1,3]],
    [[1,4],[2,2],[2,3],[2,4],[3,2],[3,4]], [[3,3]],
    [[0,5],[0,6],[0,7],[1,6],[1,7]],
    [[0,8],[1,8],[2,8],[2,7]],
    [[3,0],[4,0],[5,0],[4,1],[5,1],[6,1]],
    [[6,0],[7,0],[8,0]], [[3,7],[3,8]],
    [[1,5],[2,5],[2,6],[3,6]], [[4,2],[4,3]],
    [[4,4],[4,5],[5,2],[5,3],[5,4]], [[5,5]],
    [[4,6],[5,6]], [[4,7],[4,8],[5,8]], [[6,2],[6,3]],
    [[6,4],[6,5],[7,5],[7,6]],
    [[5,7],[6,6],[6,7],[6,8],[7,8],[8,8]],
    [[7,1],[7,2]],
    [[7,3],[7,4],[8,1],[8,2],[8,3],[8,4]],
    [[7,7],[8,7],[8,6],[8,5]],
]

solution = solve_block_party(grid)
```