Vamshi Jandhyala

The Monkey Puzzle

✠

January 2025

A combinatorics puzzle on an $n \times n$ grid.

Problem

A monkey fills a 3×3 grid with the numbers $1, 2, \ldots, 9$, one per cell. A cat writes down the three products of the rows; a dog writes down the three products of the columns. Can the monkey arrange the grid so that the cat's list and the dog's list are the same multiset?

More generally, for which n can the integers $1, 2, \ldots, n^2$ be placed in an $n \times n$ grid so that the multiset of row products equals the multiset of column products?

An impossibility bound

There are n row products and n column products. If the multisets are to coincide, each row product must equal some column product, in a bijection. The cell sitting at the intersection of paired row r and column c is the only cell common to that row and that column; it contributes to both products simultaneously. There are exactly n such cells.

Now suppose $p > n^2/2$ is a prime contained in some row product. Then p appears in the prime factorisation of exactly one cell value (since $2p > n^2$ is too large to appear). For a column product to share this prime factor, the cell containing p must lie in that column. So p forces the row's paired column to be the column that holds p. There are at most n shared cells, hence room for at most n such constraints.

Proposition. If the number of primes in $(n^2/2,\, n^2)$ exceeds n, then no solution to the puzzle exists.

Let $P(n)$ denote the count of primes in $(n^2/2,\, n^2)$.

n	3	4	5	6	7	8	9	10
$P(n)$	2	2	4	4	6	7	10	10

The bound rules out $n = 9, 10$ outright (and many larger n). For small n where $P(n) \leq n$ the bound is silent, and we must search.

Integer programming formulation

For $n = 3$ and $n = 4$ the puzzle does have a solution. Where the impossibility bound is silent, we cast the puzzle as an integer program. Let $x_{i,j,k} \in \{0, 1\}$ indicate that cell (i, j) holds the value k, and let $k = \prod_{p \in \mathbb{P}} p^{m_{k,p}}$ be the prime factorisation of k.

$$\sum_k x_{i,j,k} = 1 \qquad \text{for all } i, j \qquad \text{(each cell holds one value)}$$

$$\sum_{i,j} x_{i,j,k} = 1 \qquad \text{for all } k \qquad \text{(each value used once)}$$

$$\sum_{j,k} m_{k,p}\, x_{t,j,k} = \sum_{i,k} m_{k,p}\, x_{i,t,k} \qquad \text{for all } t, p$$

(paired row t and column t agree prime by prime)

The third constraint family says that, in the chosen pairing where row t is matched to column t, the multiplicities of every prime p in row t and column t are equal. Once that holds for every prime, the row product equals the column product.

Solutions

For $n = 3$, one valid grid is:

9	4	6	**216**
3	2	7	**42**
8	5	1	**40**
216	**40**	**42**	

For $n = 4$:

9	4	14	16	**8064**
13	3	2	12	**936**
8	5	15	7	**4200**
1	11	10	6	**660**
936	**660**	**4200**	**8064**	

Python code

```
from ortools.sat.python import cp_model
import math, itertools

def get_prime_factorisation(n):
    factors, d = {}, 2
    while n > 1:
        c = 0
        while n % d == 0:
            c += 1
            n //= d
        if c > 0:
            factors[d] = c
        d += 1
        if d * d > n:
            if n > 1:
                factors[n] = 1
```

```
            break
    return factors

def solve_grid(n):
    model = cp_model.CpModel()
    number_factors = {k: get_prime_factorisation(k) for k in range(1, n * n + 1)}
    all_primes = sorted(set().union(*number_factors.values()))

    x = {(i, j, k): model.NewBoolVar(f'x_{i}_{j}_{k}')
         for i, j, k in itertools.product(range(n), range(n), range(1, n * n + 1))}

    for i, j in itertools.product(range(n), range(n)):
        model.Add(sum(x[i, j, k] for k in range(1, n * n + 1)) == 1)
    for k in range(1, n * n + 1):
        model.Add(sum(x[i, j, k]
                      for i, j in itertools.product(range(n), range(n))) == 1)

    for t in range(n):
        for p in all_primes:
            row_sum = sum(number_factors[k].get(p, 0) * x[t, j, k]
                          for j in range(n) for k in range(1, n * n + 1))
            col_sum = sum(number_factors[k].get(p, 0) * x[i, t, k]
                          for i in range(n) for k in range(1, n * n + 1))
            model.Add(row_sum == col_sum)

    solver = cp_model.CpSolver()
    if solver.Solve(model) in (cp_model.OPTIMAL, cp_model.FEASIBLE):
        grid = [[0] * n for _ in range(n)]
        for i, j, k in itertools.product(range(n), range(n), range(1, n * n + 1)):
            if solver.Value(x[i, j, k]):
                grid[i][j] = k
        return grid
    return None
```