Finding a Feasible Solution to the "No Three in a Line Problem" by Integer Programming

IE311

Emre Kırmızı
32364

May 29, 2024

Integer Programming Formulation

Variables

Define binary variables $x_{i,j}$ where:

$$x_{i,j} = \begin{cases} 1 & \text{if a dot is placed at position } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

for $i, j = 0, 1, 2, \ldots, n-1$.

Objective Function

The objective is to maximize the total number of dots placed on the grid:

$$\text{Maximize} \quad Z = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_{i,j}$$

Constraints

No Three Collinear Dots Constraint

To ensure no three dots are collinear, the following constraints are set for each cell (a, b), for all computed slopes (i, j) (how they are computed is will be discussed below), and for all k where $k = 0, 1, 2, \ldots, n-1$ such that indices remain within grid bounds:

$$\sum_{k=0}^{n-1} x_{a+i\cdot k, b+j\cdot k} \leq 2 \quad \text{for all valid } (a + i \cdot k), (b + j \cdot k)$$

$$\sum_{k=0}^{n-1} x_{a-i\cdot k, b+j\cdot k} \leq 2 \quad \text{for all valid } (a - i \cdot k), (b + j \cdot k)$$

This ensures no line formed by any set of three dots (in all considered directions and slopes) contains more than two dots.

Computation of Slopes

Initial Approach

Initially, to avoid placing three collinear dots, we consider all possible pairs of integers (i, j) from 1 to $n-1$ as potential slopes for lines passing through any two dots. Each pair represents the slope $\frac{i}{j}$ (or $\frac{j}{i}$) of a line, and we include all these pairs directly:

Add each (i, j) pair to Slopes

This approach also incorporates special cases for vertical and horizontal lines, represented by slopes (0, 1) and (1, 0), respectively.

Efficiency Improvement

To enhance efficiency and reduce computational redundancy, we refine the slope computations by excluding redundant pairs. Redundant slopes occur when different pairs represent the same slope, e.g., $(2, 4)$ and $(1, 2)$ both represent the slope 2:

$$\text{For each } i, j = 1, \ldots, n-1, \quad \text{if } \frac{i}{j} \notin \text{SlopeSet},$$

$$\text{then add } (i, j) \text{ to Slopes and add } \frac{i}{j} \text{ to SlopeSet}$$

By storing each unique slope only once in the SlopeSet, we prevent the addition of multiple constraints for the same slope, optimizing the performance of the model.

More Efficiency Improvement

In addition to excluding redundant pairs that represent the same slope, the model introduces a further refinement by limiting the ratios considered based on the grid size n. Specifically, the model excludes pairs where the ratio $\frac{i}{j}$ or its inverse $\frac{j}{i}$ exceeds half the size of the grid:

$$\text{For each } i, j = 1, \ldots, n-1, \quad \text{if } \frac{i}{j} \notin \text{SlopeSet and}$$

$$\left(\frac{i}{j} \leq \frac{n}{2} \text{ and } \frac{j}{i} \leq \frac{n}{2}\right), \quad \text{then add } (i, j) \text{ to Slopes and add } \frac{i}{j} \text{ to SlopeSet}$$

This criterion ensures that the slopes are within a realistic range, further reducing the complexity of the problem by eliminating slopes that would lead to overly aggressive constraints, since a slope higher than $\frac{n}{2}$ (or less than $\frac{2}{n}$ as well) could not have 3 points on it in a $n \times n$ grid. This additional filtering step optimizes the constraints' processing and further enhances the model's overall performance.

Further Optimizations

Explanation for Excluding Certain Negative Slopes

In the constraints, we only consider slopes (i, j) and $(-i, j)$ but not $(-i, -j)$ or $(-i, +j)$. The reason is that if a line formed by three dots is prevented from aligning along the direction i/j and $-i/j$ for any row or column, then the same holds true for the opposite directions $-i/-j$ and $+i/-j$ automatically. This is due to the symmetry in line direction handling in the grid: if a certain set of points cannot align in one direction, they inherently cannot align in the exact opposite direction. Thus, checking one direction suffices, and including both would be redundant and computationally inefficient.