Minimum Array Partition

Daniil Grbić, Ivan Gogić

September 29, 2023

1 Problem description

Given a $n \times n$ array A of non-negative integers, and a positive integer p, we need to choose $p-1$ horizontal dividers $0 = h_0 < h_1 < \cdots < h_p = n$ and $p-1$ vertical dividers $0 = v_0 < v_1 < \cdots < v_p = n$ partitioning A into p^2 blocks, so as to minimize the following expression:

$$\max_{\substack{1 \le i \le p \\ 1 \le j \le p}} \sum_{\substack{v_{i-1} \le x \le v_i \\ h_{j-1} \le y \le h_j}} A[x, y]$$

2 Methods

2.1 Brute Force (BF)

Firstly, we write a brute force algorithm for the purpose of finding optimal solutions for small test cases, which we will use to evaluate our optimisation algorithms.

BF generates all possible partitions of the matrix, and chooses the best one of them. This essentially means that BF will need an unreasonable amount of time to solve all but the smallest of testcases.

The complexity of this approach is $O(n, p) = \binom{n}{p}^2 * p^2$, indeed $\binom{n}{p}^2$ is the total number of partitions, while for each of them we need to check p^2 blocks. We find the sum of each block in constant time, by utilising a partial sum DP matrix.

2.1.1 Cutting optimization

A small optimization of the previous approach involves abandoning some partitions early when it is clear that they will not yield a better solution than the current known best.

Consider this: we know the sum S of the entire matrix M, and we know there are p^2 blocks in total. According to the Pigeonhole principle, at least one of said blocks will have a sum greater or equal to $\frac{S}{p^2}$.

Alternatively, if we have checked $m < p^2$ blocks and their total sum is $S_m < S$, then the maximum value of the remaining blocks will be greater or equal to $\frac{S-S_m}{m-p^2}$.

If at any point this sum happens to become greater than the best solution we have found beforehand, we can safely skip checking the rest of the partition in question. In practice, this speeds up the algorithm 3 to 5 times.

2.2 Genetic Algorithm (GA)

A genetic algorithm (GA) is a p-metaheuristic inspired by the process of natural selection. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection.

We use a nearly textbook GA implemenation. Nevertheless, given the specificity of the problem, crossover and mutation implementation deserve a closer look.

2.2.1 Crossover

Crossover of two partitions can be done by an arithmetical approach. For simplicity, we will explain the process on a one-dimensional partition. This can be directly applied to two dimensions for the problem in question.

Consider two individuals, A and B, with partitions $[1, 3, 9]$ and $[2, 5, 7]$. Their child C will try to take a middle path in the hopes of bettering their fitness, by taking the average of respective divider pairs: $C = [\frac{1+2}{2}, \frac{3+5}{2}, \frac{9+7}{2}] = [1.5, 4, 8]$. Since dividors have to be whole numbers, we fix C by either rounding the first element down, or rounding it up (chosen randomly).

2.2.2 Mutation

Mutation is achieved by trying to make small movements of individua dividers with a given probability to do so, considering there is an available place to move.

2.3 Simulated Annealing (SA)

2.4 Variable Neighbourhood Search (VNS)

3 Method evaluation

3.1 Test generation

We wrote a helper script for generating testcases. The script takes 3 command line arguments: n, p, and *type*; where *type* refers to how the matrix M is generated:

Ones fills M with ones, mainly used for debugging

Random fills the matrix with numbers from $(1, n/2)$

Linear , $M_{i,j} = i + j + 1$

Squared , $M_{i,j} = i^2 + j^2 + 1$

3.2 Results

All 4 methods are evaluated on a set of 10 representative testcases. Each algorithm is run 5 times on each prolem insatnce, and we record the best, worst, and average results and execution times.

Let us begin with testcases representing the *random* type. For this type of problems, we can see that SA finds close to optimal solutions at a fraction of the time:

```
Testcase: tests/random_n50_p3.in
  BRUTE FORCE
    Results  : BEST=3769       WORST=3769       AVG=3769.00
    Time (s) : BEST=2.35       WORST=2.55       AVG=2.45
  GENETIC ALGORITHM (GA) [pop_size=50, num_iters=250, elitism_size=10]
    Results  : BEST=3769       WORST=3769       AVG=3769.00
    Time (s) : BEST=0.81       WORST=0.81       AVG=0.81
  SIMULATED ANNEALING (SA)
    Results  : BEST=3769       WORST=3769       AVG=3769.00
    Time (s) : BEST=0.13       WORST=0.14       AVG=0.14
  VARIABLE NEIGHBOURHOOD SEARCH (VNS) [num_iters=2500]
    Results  : BEST=3769       WORST=3769       AVG=3769.00
    Time (s) : BEST=0.47       WORST=0.48       AVG=0.47

Testcase: tests/random_n50_p6.in
  BRUTE FORCE - skipping due to speed
  GENETIC ALGORITHM (GA) [pop_size=50, num_iters=250, elitism_size=10]
```

```
    Results  : BEST=1043       WORST=1046       AVG=1045.40
    Time (s) : BEST=1.62       WORST=1.68       AVG=1.65
  SIMULATED ANNEALING (SA)
    Results  : BEST=1043       WORST=1046       AVG=1043.60
    Time (s) : BEST=0.97       WORST=1.04       AVG=1.00
  VARIABLE NEIGHBOURHOOD SEARCH (VNS) [num_iters=2500]
    Results  : BEST=1043       WORST=1046       AVG=1043.60
    Time (s) : BEST=3.18       WORST=3.44       AVG=3.28

Testcase: tests/random_n100_p5.in
  BRUTE FORCE - skipping due to speed
  GENETIC ALGORITHM (GA) [pop_size=100, num_iters=500, elitism_size=20]
    Results  : BEST=11024      WORST=11066      AVG=11033.40
    Time (s) : BEST=5.52       WORST=5.71       AVG=5.63
  SIMULATED ANNEALING (SA)
    Results  : BEST=11024      WORST=11066      AVG=11037.20
    Time (s) : BEST=0.60       WORST=0.66       AVG=0.64
  VARIABLE NEIGHBOURHOOD SEARCH (VNS) [num_iters=10000]
    Results  : BEST=11024      WORST=11066      AVG=11049.20
    Time (s) : BEST=9.29       WORST=9.43       AVG=9.39
```

Next, let us compare the results achieved on the *linear* test type.

```
Testcase: tests/linear_n20_p4.in
  BRUTE FORCE
    Results  : BEST=602        WORST=602        AVG=602.00
    Time (s) : BEST=4.71       WORST=5.17       AVG=4.99
  GENETIC ALGORITHM (GA) [pop_size=20, num_iters=100, elitism_size=4]
    Results  : BEST=602        WORST=602        AVG=602.00
    Time (s) : BEST=0.17       WORST=0.18       AVG=0.17
  SIMULATED ANNEALING (SA)
    Results  : BEST=602        WORST=602        AVG=602.00
    Time (s) : BEST=0.32       WORST=0.34       AVG=0.33
  VARIABLE NEIGHBOURHOOD SEARCH (VNS) [num_iters=400]
    Results  : BEST=602        WORST=608        AVG=603.20
    Time (s) : BEST=0.18       WORST=0.20       AVG=0.19

Testcase: tests/linear_n30_p4.in
  BRUTE FORCE - skipping due to speed
  GENETIC ALGORITHM (GA) [pop_size=30, num_iters=150, elitism_size=6]
    Results  : BEST=1920       WORST=1968       AVG=1958.40
    Time (s) : BEST=0.37       WORST=0.39       AVG=0.38
  SIMULATED ANNEALING (SA)
    Results  : BEST=1920       WORST=1920       AVG=1920.00
    Time (s) : BEST=0.32       WORST=0.35       AVG=0.34
  VARIABLE NEIGHBOURHOOD SEARCH (VNS) [num_iters=900]
```

```
    Results  : BEST=1920      WORST=1968      AVG=1939.20
    Time (s) : BEST=0.43      WORST=0.44      AVG=0.43
```

We find the results for *squared* test the most intersting.

```
Testcase: tests/squared_n40_p3.in
  BRUTE FORCE
    Results  : BEST=214662    WORST=214662    AVG=214662.00
    Time (s) : BEST=1.60      WORST=1.72      AVG=1.65
  GENETIC ALGORITHM (GA) [pop_size=40, num_iters=200, elitism_size=8]
    Results  : BEST=214662    WORST=215757    AVG=215538.00
    Time (s) : BEST=0.52      WORST=0.53      AVG=0.52
  SIMULATED ANNEALING (SA)
    Results  : BEST=214662    WORST=214662    AVG=214662.00
    Time (s) : BEST=0.13      WORST=0.13      AVG=0.13
  VARIABLE NEIGHBOURHOOD SEARCH (VNS) [num_iters=1600]
    Results  : BEST=214662    WORST=215757    AVG=215538.00
    Time (s) : BEST=0.30      WORST=0.31      AVG=0.30

Testcase: tests/squared_n40_p4.in
  BRUTE FORCE - skipping due to speed
  GENETIC ALGORITHM (GA) [pop_size=40, num_iters=200, elitism_size=8]
      Results  : BEST=124440    WORST=131775    AVG=125907.00
      Time (s) : BEST=0.66      WORST=0.69      AVG=0.68
  SIMULATED ANNEALING (SA)
      Results  : BEST=124440    WORST=124440    AVG=124440.00
      Time (s) : BEST=0.31      WORST=0.33      AVG=0.32
  VARIABLE NEIGHBOURHOOD SEARCH (VNS) [num_iters=1600]
      Results  : BEST=124440    WORST=131775    AVG=125907.00
      Time (s) : BEST=0.76      WORST=0.81      AVG=0.78

Testcase: tests/squared_n100_p10.in
  BRUTE FORCE - skipping due to speed
  GENETIC ALGORITHM (GA) [pop_size=100, num_iters=500, elitism_size=20]
    Results  : BEST=858130    WORST=984543    AVG=941758.40
    Time (s) : BEST=14.10     WORST=14.69     AVG=14.47
  SIMULATED ANNEALING (SA)
    Results  : BEST=903805    WORST=1149356   AVG=1030165.60
    Time (s) : BEST=4.56      WORST=4.85      AVG=4.69
  VARIABLE NEIGHBOURHOOD SEARCH (VNS) [num_iters=10000]
    Results  : BEST=837551    WORST=928305    AVG=868049.20
    Time (s) : BEST=28.35     WORST=33.75     AVG=30.67
```

4 Failed ideas

5 Conclusion

6 References