Simulated Annealing for the Construction of Hadamard Matrices

Raven Ruiz and Adanary Ramirez
Faculty Advisor: Dr. Andras Balogh

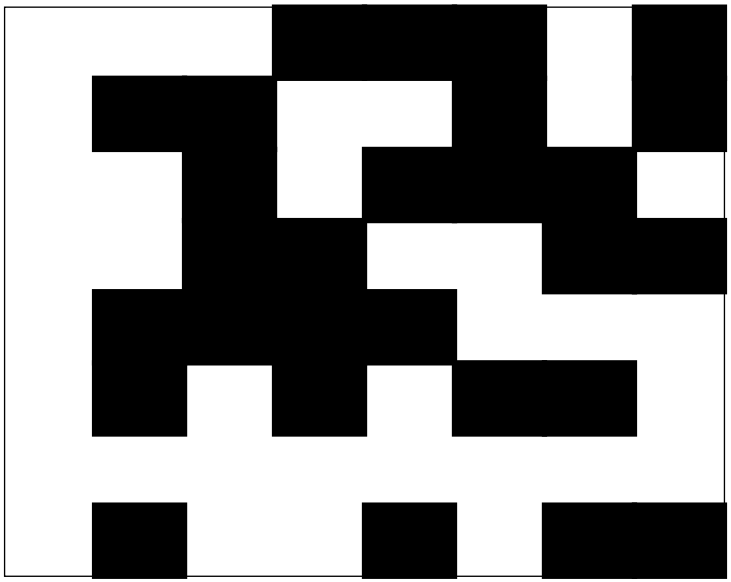
University of Texas Rio Grande Valley
raven.ruiz01@utrgv.edu

November 20, 2020

Definition

- Hadamard Matrices are $m \times m$ square matrices with entries $+1$ and -1 and with mutually orthogonal columns. The rows are also mutually orthogonal.

$$\begin{pmatrix} + & + & + & - & - & - & + & - \\ + & - & - & + & + & - & + & - \\ + & + & - & + & - & - & - & + \\ + & + & - & - & + & + & - & - \\ + & - & - & - & - & + & + & + \\ + & - & + & - & + & - & - & + \\ + & + & + & + & + & + & + & + \\ + & - & + & + & - & + & - & - \end{pmatrix}$$



History and Applications

- Hadamard Matrices were first discovered by James Sylvester in 1867 who established how to create matrices of order 2^k for any $k \in \mathbb{N}$.
- In 1893, Jacques Hadamard introduced the Hadamard Conjecture, stating that an Hadamard matrix exists when $m = 1$, $m = 2$, and $m = 4k$ where $k \in \mathbb{N}$.
- The smallest multiple of 4 (but not of the form 2^k) for which no Hadamard matrix is known is 668.
- Hadamard matrices are used in applications such as image processing and error-correcting codes.

Important Property of Hadamard Matrices

$$Q^T Q = \begin{pmatrix} m & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & m \end{pmatrix}$$

$$E(Q) = \sum_{i,j} |Q^T Q| - m^2 = 0$$

- We will use $E(Q)$ as energy to measure how close a matrix is to being an Hadamard matrix.

Simulated Annealing

- We were inspired by A. Suksmono's work that uses a method called The Simulated Annealing Algorithm with a Metropolis update criteria on an ising model in order to construct Hadamard matrices.
- The Simulated Annealing is a stochastic algorithm to find global minimum of a function.
- This process is similar to the annealing of metals where the metal is heated up and then slowly cooled down in order to reduce its hardness.
- The algorithm requires large number of matrix algebra operations, which can be very slow when the computation is done in serial way.
- We build on a previous math project by Adanary Ramirez, who achieved partial parallelization of the algorithm.

Simulated Annealing Algorithm

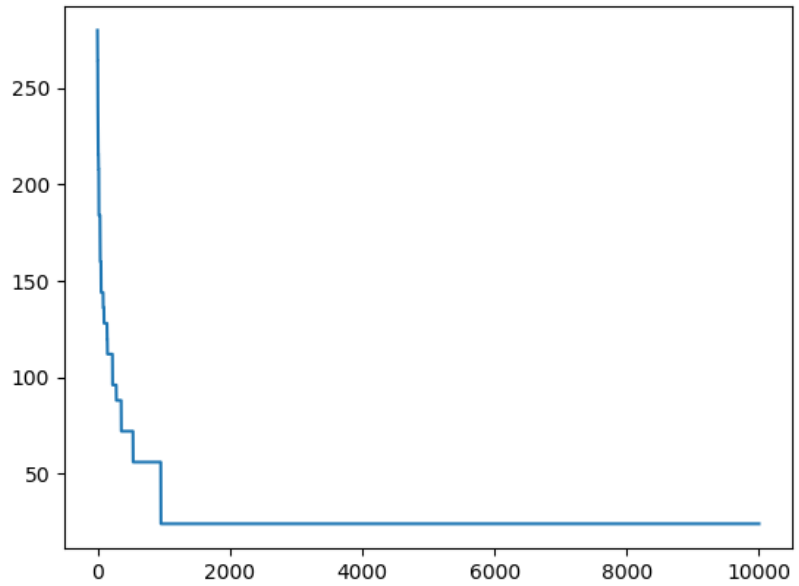
- Start by randomly selecting a Q_0 matrix with balanced $+1$ and -1 entries in each column except for the first one, which consists of all $+1$.
- For a matrix Q we define its energy as

$$E(Q) = \sum_{i,j} |Q^T Q| - m^2$$

- While E(Q)$>$ 0 we randomly flip $+1$ and -1 entries from random columns
  - If the energy decreases then we accept the change and accept the new Q matrix.

Problem with the Algorithm

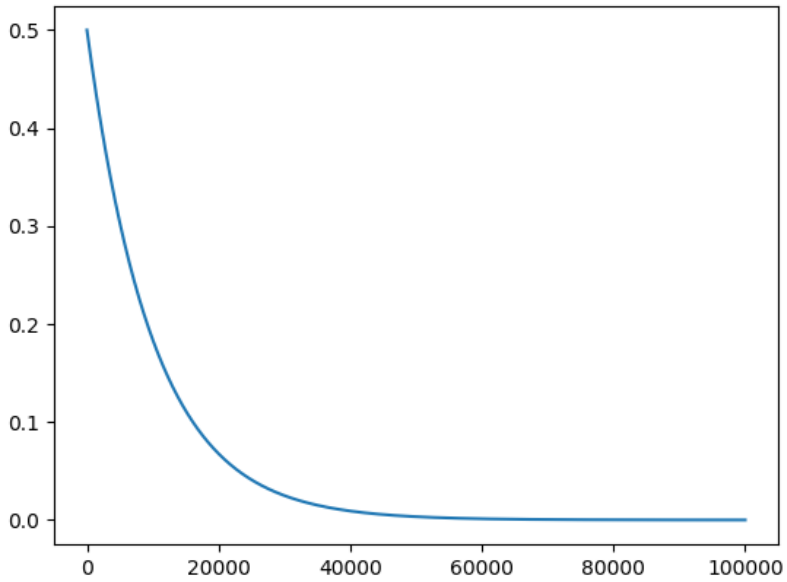
- The algorithm can get stuck in a bad matrix configuration that does not improve anymore



- A solution to this problem is The Metropolis - Hasting method.

Metropolis-Hasting Update Criteria

Probability as a Function of Time t: $P = 0.5e^{-Ct}$



- Accept the new matrix with some probability even if its energy is not smaller in order to avoid getting stuck in a bad configuration.
- If C is too large, then the process might get stuck.
- If C is too small, then the convergence is too slow.

Improvement - Fixing Columns

- Depending on the size of the matrix, we were able to fix a certain number of columns by rearranging rows, such that these columns are already mutually orthogonal to each other.
- This means the algorithm has less columns to work with.
- For $k = 2$, This resulted in a 40% decrease in iteration steps on average.

$$\begin{pmatrix} + & + & + & - & - & - & + & - \\ + & - & - & + & + & - & + & - \\ + & + & - & + & - & - & - & + \\ + & + & - & - & + & + & - & - \\ + & - & - & - & - & + & + & + \\ + & - & + & - & + & - & - & + \\ + & + & + & + & + & + & + & + \\ + & - & + & + & - & + & - & - \end{pmatrix} \quad \begin{pmatrix} + & + & + & + & + & + & + & + \\ + & + & + & - & - & - & + & - \\ + & + & - & + & - & - & - & + \\ + & + & - & - & + & + & - & - \\ + & - & + & + & - & + & - & - \\ + & - & + & - & + & - & - & + \\ + & - & - & + & + & - & + & - \\ + & - & - & - & - & + & + & + \end{pmatrix}$$

Fixing Columns $k = 3$

- This resulted in a 10% decrease in iteration steps on average.

$$\begin{pmatrix}
+ & + & + & - & - & - & + & - & + & - & + & + \\
+ & + & + & + & + & + & - & + & - & - & + & + \\
+ & + & + & - & - & + & + & + & - & + & - & - \\
+ & + & - & - & + & - & - & - & - & + & + & - \\
+ & + & - & + & - & - & - & + & + & - & - & - \\
+ & + & - & + & + & + & + & - & + & + & - & + \\
+ & - & + & + & + & - & + & + & + & + & + & - \\
+ & - & + & + & - & - & - & - & - & + & - & + \\
+ & - & + & - & + & + & - & - & + & - & - & - \\
+ & - & - & - & + & - & + & + & - & - & - & + \\
+ & - & - & + & - & + & + & - & - & - & + & - \\
+ & - & - & - & - & + & - & + & + & + & + & +
\end{pmatrix}$$

CPU vs GPU

- CPUs do mostly serial calculations and working with large matrices is ineffective on them.
- GPUs (graphics processing units) can do thousands of calculations in parallel.
- In the previous project some but not all steps were done on the GPU and the frequent data transfer between CPU and GPU slowed down the calculations.
- In this project we implemented the flipping of $+1/-1$ pairs on the GPU using the Python package CuPy.

Working on the GPU

- Create balanced number of random $+1$ and -1 entries in each column

```
for J in range(s,m):
    Q[:, J] =
        cupy.sign(cupy.random.permutation(m)-(2*k-0.5))
```

- Calculate energy: $E = \sum_{i,j} \left| Q^T Q \right|_{ij} - m^2$

```
E[0] =
    cupy.sum(cupy.absolute(cupy.dot(cupy.transpose(Q),Q)))-m2
```

- Copy matrix Q to $Q1$

```
Q1=cupy.copy(Q)
```

CUDA Kernel Code in Python

```
modules = SourceModule("""
__global__ void flip_pairs(float *Q, int col, int row1,
                           int row2, int M)
{
    int i = blockDim.x*blockIdx.x+threadIdx.x;
    int i1 = row1*M+col;
    int i2 = row2*M+col;
    //If index match and signs are opposite, flip signs
    if(((i==i1) || (i==i2)) && (Q[i1] != Q[i2]))
    {
        Q[i1] *= -1.0; // Flip signs
        Q[i2] *= -1.0;
    }
}
""")
blocksx = 64
blocks = (blocksx,1,1)
grids = (math.ceil(m2/blocksx),1,1)
Flip_Pairs(grids, blocks, (Q1, colindx, rowindx1, rowindx2, m))
```

Speed Comparison of the Kernel code for 10^5 Iteration

Size of Matrix	**Serial Code**	**CuPy Code**	**Speed up from Serial (times)**
20×20	9s	32s	0.3
40×40	12s	32s	0.4
100×100	61s	32s	1.9
500×500	7861s	57s	137.9
668×668	18600s	94s	197.9

- For small matrices our CuPy code was slower than the serial code.
- For the 500×500 matrix our CuPy code was more than $100\times$ faster than the serial code.
- For the 668×668 matrix our CuPy code was almost $200\times$ faster than the serial code.

Summary and Future Plans

- In order to find Hadamard matrices using stochastic computations (Simulated Annealing):
  - We made all computational steps parallel on the GPU instead of on the CPU.
  - We fixed some columns to reduce the number of columns that are needed to change.
  - The largest Hadamard found so far is a 16×16 matrix.
- Future Plans
  - Automate the fixing of columns for larger matrices.
  - Change the probability function for the Metropolis-Hasting update criteria to automate the avoidance of getting stuck.
  - Find a previously unknown Hadamard matrix (size 668 matrix).

References

Suksmono, Andriyan (2016)

Finding a Hadamard Matrix by Simulated Annealing of Spin-Vectors

Journal of Physics: Conference Series 18(3), 66 – 70.

Suksmono, Andriyan (2016)

Probabilistic Construction and Analysis of Seminormalized Hadamard Matrice

arXiv:1606.09368 15(2), 6–12.

Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido and Crissman Loomis.

CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations.

Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS), (2017).