FINDING HADAMARD AND (ε, δ)-QUASI-HADAMARD MATRICES WITH OPTIMIZATION TECHNIQUES

By
Samuel Buteau, B.Sc.
August 2016

A Thesis
submitted to the School of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Master of Science in Mathematics[1]

[1]The M.Sc. Program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

Abstract

Existence problems (proving that a set is nonempty) abound in mathematics, so we look for generally applicable solutions (such as optimization techniques). To test and improve these techniques, we apply them to the Hadamard Conjecture (proving that Hadamard matrices exist in dimensions divisible by 4), which is a good example to study since Hadamard matrices have interesting applications (communication theory, quantum information theory, experiment design, etc.), are challenging to find, are easily distinguished from other matrices, are known to exist for many dimensions, etc..

In this thesis we study optimization algorithms (Exhaustive search, Hill Climbing, Metropolis, Gradient methods, generalizations thereof, etc.), improve their performance (when using a Graphical Processing Unit), and use them to attempt to find Hadamard matrices (real, and complex). Finally, we give an algorithm to prove non-trivial lower bounds on the Hamming distance between any given matrix with elements in $\{+1, -1\}$ and the set of Hadamard matrices, then we use this algorithm to study matrices with similar properties to Hadamard matrices, but which are far away (with respect to the Hamming distance) from them.

Acknowledgements

I would like to thank my supervisors, and both the NSERC and the Ontario Graduate Scholarship programs.

Dedication

À papa et maman.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We say that a real-valued $(m \times m)$-matrix M is a (real) *Hadamard* matrix if its coefficients satisfy

$$[M]_{ij} \in \{-1, +1\} \tag{1}$$

and if

$$MM^T = m\mathbb{I} \tag{2}$$

where M^T is the transpose of M and $\mathbb{I}$ the $(m \times m)$-identity matrix.

We will denote by $\mathcal{H}_m$ the set of (real) Hadamard matrices of size m. Clearly, $\mathcal{H}_1$ and $\mathcal{H}_2$ are non-empty. Indeed,

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \in \mathcal{H}_2$$

had already been considered by Sylvester in 1867.

In 1893, J. Hadamard proved (see Proposition 4.1.3) that for $m \geq 4$, if $\mathcal{H}_m \neq \emptyset$ then 4 divides m, and conjectured that the converse also holds. This conjecture is still open; the first unsolved case is $m = 668$. Furthermore, both

$$\{m \in \mathbb{N} \,|\text{Hadamard conjecture is open}\}$$

and

$$\{m \in \mathbb{N} \,|\text{Hadamard conjecture is solved}\}$$

are infinite sets.

Once we have shown $\mathcal{H}_m \neq \emptyset$ for a given m, we consider the problem of explicitly listing all elements of $\mathcal{H}_m$. This problem is solved for all $m \leq 32$, and unsolved for all $m > 32$, see [25].

Central to both the proof that $\mathcal{H}_m \neq \emptyset$ and the enumeration of the elements of $\mathcal{H}_m$ is the problem of constructing a Hadamard matrix. Many such constructions are known, yet they are invariably *ad hoc*. Therefore, we look for a method to find $H \in \mathcal{H}_m$ whenever $\mathcal{H}_m \neq \emptyset$ and which works for every m.

In theory, we know that such a method exists, since the matrices which satisfy Eq. (1) form a finite set (for a given m) and can be easily enumerated. Therefore, the following strategy should eventually yield a Hadamard matrix if $\mathcal{H}_m \neq \emptyset$:

1. Enumerating $(m \times m)$-matrices M with $[M]_{ij} \in \{-1, +1\}$ for $1 \leq i, j \leq m$ one by one.

2. Testing if $MM^T = m\mathbb{I}$.

3. Terminating when we find the first Hadamard matrix.

However, there are 2^{m^2} $(m \times m)$-matrices satisfying Eq. (1), and so this method (exhaustive enumeration) quickly becomes impractical. (We have $2^{m^2} > 10^{10^5}$ when $m \geq 668$.)

We must therefore look for more sophisticated methods. There is a simple way in which Hadamard matrices can be distinguished from matrices only satisfying Eq. (1). For instance, we can see that the following function f assigns to them a value of 0 while it assigns a strictly positive value to any other matrix satisfying Eq. (1):

$$f(M) = \sum_j \sum_{i<j} \left(\left[MM^T \right]_{ij} \right)^2$$

If we denote by X the set of 2^{m^2} $(m \times m)$-matrices satisfying Eq. (1), then X, f, and $y = 0$ are sufficient to recover $\mathcal{H}_m$. This allows us to discuss optimization methods. In general, optimization methods will try to minimize (or maximize) a function f (in this case, find an element of $\mathcal{H}_m$) by iteratively apply the following steps:

1. Evaluate f at some points in X,
2. Based on the result, decide on which new points to re-evaluate f.
3. Repeat until we run out of resources (time, space, energy, money, etc.) or we find element of $\mathcal{H}_m$.

In this thesis, we have considered such methods. In particular, we have asked what are the fundamental factors which determine whether we succeed or fail in finding an element of $\mathcal{H}_m$ with a given amount of resources. Here are some main factors:

- The decisions that we make,
- The function f that we use to optimize, and
- The number of evaluations we can perform using our fixed resources.

The first two factors related to the behavior of the methods while the last relates to the speed with which it performs this behavior. In Chapters 3,4, and 5, we have made improvements to all three factors, thus coming up with powerful optimization algorithms for finding Hadamard matrices, and what we call (ε, δ)-Quasi-Hadamard matrices.

This brings us to the question of why we apply optimization techniques to find Hadamard matrices as opposed to any other open problem in mathematics. Indeed, such methods could be applied to many interesting problems. However, the fact that we can easily know if a given matrix is a Hadamard matrix, combined with the fact that we know such matrices exist for many values of m provides us with a test-bed in which to develop new optimization methods, and understand the limits of optimization.

Furthermore, Hadamard matrices are interesting in their own right, and despite many applications, such as in error-correcting codes, we still do not know if any such matrices exist for $m = 668$.

After having developed these powerful optimization methods, we used them to study objects related to Hadamard matrices.

Indeed, even if f has the Hadamard matrices as its global minima, the methods do not always find the smallest possible value for f. Instead, we get matrices M with $f(M)$ ε-close to the global minimum of 0, for some positive ε. Empirically, we have found that even by imposing a relatively small value for ε, these matrices M could still be δ-far from the Hadamard matrices. More precisely, we would get M such that

$$\min_{H \in \mathcal{H}_m} d_{Hamming}(M, H) \geq \delta$$

where $d_{Hamming}(M, M') = \left| \left\{ (i,j) \,\middle|\, [M]_{ij} \neq [M']_{ij} \right\} \right|$ is the Hamming distance. Such matrices are called (ε, δ)-Quasi-Hadamard matrices, and we discuss their properties in the thesis.

Here is a detailed description of the content of the thesis.

- In Chapter 2, we present the notation used and give some background necessary to define algorithms, prove their behavior, and measure their performance, in a mathematical context.

- In Chapter 3, we formalize questions of optimization, give basic methods to optimize, generalize such methods and propose the methods we use in subsequent chapters. We also show a way to sample from a distribution with finite support as a reduction problem, which can be performed in parallel. Finally, we represent the proposed optimization methods in terms of finite-support sampling.

- In Chapter 4, we introduce Hadamard matrices, and describe them as the preimage of the global minimum of *ad hoc* functions.

- In Chapter 5 we present actual algorithms which implement the methods proposed in Chapter 3 for the special cases of the *ad hoc* functions found in Chapter

4. We present these algorithms in the general context of complex Hadamard matrices. These algorithms achieve considerable speedups both in terms of total work and in terms of Parallelism, when compared to a naive algorithm for the same sequence $(M_i)_{i\in\mathbb{N}}$. (For instance, if we divide the number of determinants computed naively by the total number of operations actually performed, we get, depending on how we choose a stability parameter, either a linear, or a sublinear (in the dimension m) number of operations. Such technicalities are discussed precisely in the text of Chapter 5.) Finally, we analyze both the behavior and the performance of the algorithms presented.

- In Chapter 6, we study (ε, δ)-Quasi-Hadamard matrices, exploring why they occur and their relationship to Hadamard matrices. We also develop an algorithm to compute a lower bound on the Hamming distance between a given matrix in $\mathcal{M}_m(\pm)$ and $\mathcal{H}_m$. Finally, we give an example of such a (ε, δ)-Quasi-Hadamard matrix (obtained with the algorithms presented in Chapter 5) and prove some of its properties.

Here are the **major novel contributions** of this thesis.

1. In Chapter 3, we define the odds combination function, which allows parallel sampling of finitely-supported distributions. Unless otherwise stated in the text, most of the ideas of Chapter 3 are not new, but the unified discussion of the various techniques of optimization and their comparison is novel to this thesis.

2. In Chapter 5, we define six algorithms (which we call $\mathrm{FS}[\zeta][\eta]$) to find Hadamard matrices (both real and complex). We analyze their behavior, proving that it is a special case of the methods outlined in Chapter 3. We then analyze their performance, both the total number of operations required to perform them, and the impact of using multiple processors. Unless otherwise specified in the text, the original ideas are presented in Appendix B and C, but the combination of these ideas into practical algorithms is novel.

3. In Chapter 6, we give an algorithm to determine a useful lower bound on the Hamming distance between a given matrix $M \in \mathcal{M}_m(\pm)$ and $\mathcal{H}_m$. Furthermore,

we give some properties of a matrix found using the optimization algorithms developed in Chapter 5. Finally, unless otherwise stated in the text, all the technical results (lemmas, propositions, etc.) obtained in Section 6.1 while trying to understand (ε, δ)-Quasi-Hadamard matrices is novel to this thesis.

Chapter 2

Notation for Algorithms

In this thesis, we discuss various algorithms. We define them, show

1. how they function (what they do; how they modify their inputs and what outputs they produce; their *meaning* or *semantics* in short),
2. the number of operations they execute based on their inputs, and
3. how the execution time is affected in the presence of one or many processors (their parallelism).

The target audience for this thesis is primarily mathematicians, but some results might be of interest to computer scientists, software engineers, and even theoretical physicists. This means that we must aim at a presentation that is as rigorous as possible while being recognized by many disciplines, as straightforward as possible, and close enough to a programming language that the implementation requires only a basic translation.

One example of such a treatment is that of algorithms in Donald Knuth's *The Art of Computer Programming*, Volumes 1 to 4A [26, 28, 27, 29]. . Where possible, we will use the same notations.

However, Knuth's notations do not allow one to describe the parallel aspect of algorithms.

Furthermore, Knuth describes randomness within programs at a level lower than the one appropriate for this thesis.

For these reasons, we have supplemented Knuth's approach with extra annotations (of the form $[\![\cdot]\!]$ and $(\!|\cdot|\!)$) to denote the use of either parallelism or randomness, respectively.

We also give details about the model of parallelism and randomness used. There is nothing revolutionary in either of those. They are, if anything, more simple and concrete than many well-known formalizations thereof, and much simpler than actual computers. Where they differ from similar concepts in the literature, it is to simplify the presentation and unify it around the style developed by Knuth.

Finally, we introduce the bare minimum in terms of asymptotic notation, semantics, and parallel performance measures. Some of this comes again from *The Art of Computer Programming*, but also from [47]. Notably, this is the source of our notation for randomness, but also for asymptotic notation. The concepts of parallel performance measures comes from [39, 2, 1, 22], .

2.1 How to define Algorithms

Here is an example of an algorithm

Algorithm X (*Example Name*). We describe here the goal of the algorithm. Given 4 real numbers a, b, c, d, we output $\alpha_1, ..., \alpha_4 \in \mathbb{R}$ and $\beta \in \mathbb{R}^4$. After termination, we will have $\alpha_1 = a_{[in]}$, $\alpha_2 = a^2_{[in]}$, $\alpha_3 = a^4_{[in]}$, $\alpha_4 = a^8_{[in]}$, and $\beta_i = a_{[in]}\alpha_i$ for $1 \leq i \leq 4$, where the subscript $[in]$ denotes the value a variable had at the start of the algorithm. We will denote the invocation of algorithm X by $\alpha, \beta \leftarrow \text{MadeUpExample}\,(a, b, c, d)$.

X1. [First Sequential Step] Set $\alpha_1 \leftarrow a$.

X2. [Second Sequential Step] Set $\alpha_2 \leftarrow b\alpha_1$.

X3. [Third Sequential Step] Set $\alpha_3 \leftarrow c\alpha_2$.

X4. [Fourth Sequential Step] Set $\alpha_4 \leftarrow d\alpha_3$.

X5. [Fifth Sequential Step] For $i = 1, 2, 3$, set $\alpha_{i+1} \leftarrow \alpha_i\alpha_i$. (This happens sequentially, which means, for instance, that "$\alpha_3 \leftarrow \alpha_2\alpha_2$" will be evaluated using the new value of α_2, namely α_1^2, instead of the old value, namely ba. Note that step

X5, as written, makes it so that steps X2, X3, and X4 could be removed without changing the final values of the variables.)

X6. [First Parallel Step] $[\![\forall_{i\in\{1,2,3,4\}}]\!]$. set $\beta_i \leftarrow \alpha_i\alpha_1$. (This happens in parallel or sequentially, and unless we explicitly say otherwise, this symbol implies that any choice for the order in which these four operations occur will be interchangeable. Namely, if the values of all variables are prepared identically for two orderings of the four operations, then after all operations have been executed, the final values of the variables will be the same in each case.)

X7. [Termination step] Terminate. (We will omit the termination step when it is the last one. But we will always mention it when it happens elsewhere.) ∎

Note the short description in brackets at the beginning of each step, and the occasional text in parenthesis at the end of each step. These are comments, and not part of the algorithm. This follows the convention from [26, Vol 1, p.2-4].

Remark 2.1.1. Sometimes, we will need to use randomness in our algorithms. Of course, on a computer, such randomness is implemented in terms of deterministic operations (pseudo-randomness), but at the level of algorithms, it is easier to understand the intent if we discuss randomness in terms of random variables. However, to conform with the way implementations actually work, we will use the convention that whenever the instruction Variable $\leftarrow (\!|\text{RandomVariable}|\!)$ appears, Variable is to receive a value distributed according to RandomVariable. However, all instances of random variables bracketed by $(\!|\cdot|\!)$ within a program are taken to be independent of each other. In other words

$$\begin{aligned} a &\leftarrow (\!|X|\!) \\ b &\leftarrow (\!|X|\!) \\ c &\leftarrow (\!|Y|\!) \end{aligned}$$

will put values into (a, b, c) such that a, b follow the distribution of X, and c follows the distribution of Y, but all three are independent. Conceptually, we can imagine that for every random variable in the program, we keep a sequence of independent

and identically distributed copies, indexed by i. Then, throughout the program, we carry an extra variable i, and instead of using the random variable X, we use the copy X_i, and we make sure that every instance of "Variable $\leftarrow$ (|RandomVariable|)" uses a different value for i.[1]

2.2 The Meaning of Algorithms

Algorithms modify the values of variables. To understand how an algorithm behaves is to understand how the final values of both the input and output variables depend upon the initial values of the input variables, or in the case of a randomized algorithm, how the final *distribution* of both the inputs and the outputs depends on the input values.

Formally, if an algorithm A has inputs $I_1, ..., I_n$ and outputs $O_1, ..., O_m$ belonging to sets $T_1, ..., T_n$ and $T_{n+1}, ..., T_{n+m}$ respectively, then this dependency is encoded in a function

$$f : \prod_{i=1}^{n} T_i \to \prod_{i=1}^{n+m} T_i$$

(in the deterministic case), or a function

$$f : \prod_{i=1}^{n} T_i \to \left(\prod_{i=1}^{n+m} T_i \right)^{\Omega}$$

(in the case where randomness is present), where $\prod_{i=1}^{n} A_i = A_1 \times A_2 \times ... \times A_n$ is the

[1] This can be done quite easily when there is only one processor, simply by rewriting

$$\text{Variable} \leftarrow (|\text{RandomVariable}|)$$

as

$$\begin{aligned} \text{Variable} &\leftarrow \text{RandomVariable}_i \\ i &\leftarrow i + 1 \end{aligned}$$

with "i" being renamed if needed to avoid clash with some other variable in the algorithm. It can also be done on a computer with finitely many (p) processors, by keeping one index variable per processor, such that they begin the program with values 1 to p, and Variable $\leftarrow$ (|RandomVariable|) is rewritten as Variable $\leftarrow \text{RandomVariable}_i, i \leftarrow i + p$.

n-fold Cartesian product, and if A, Ω are sets, A^{Ω} is the set of functions from Ω with values in A (see [47, pp. 2-11]).

Namely, this function is given by

$$f\left(I_{1,[in]}, ..., I_{n,[in]}\right) = \left(I_{1,[out]}, ..., I_{n,[out]}, O_{1,[out]}, ..., O_{m,[out]}\right)$$

where $\left(I_{1,[out]}, ..., I_{n,[out]}, O_{1,[out]}, ..., O_{m,[out]}\right)$ is the final value of the input and output variables respectively obtained by invoking algorithm A with inputs having initial value $I_{1,[in]}, ..., I_{n,[in]}$. This allows us to associate to any algorithm (the sequence of operations to be executed) a function which summarizes the execution without referring to mutable variables. To reiterate, we will use the convention that marking a variable with the subscript $[in]$ or with the subscript $[out]$ will denote the value of that variable just before the algorithm begins and just after the algorithm terminates, respectively.

A statement about the behavior of an algorithm would then look something like:

Claim 2.2.1. (Example) For all real $a \in \mathbb{R}$, if the initial values

$$\left(I_{1,[in]}, ..., I_{n,[in]}\right) \in \prod_{i=1}^{n} T_i$$

satisfy "$I_{1,[in]} = a$," then the corresponding final values

$$\left(I_{1,[out]}, ..., I_{n,[out]}, O_{1,[out]}, ..., O_{m,[out]}\right) \in \prod_{i=1}^{n+m} T_i$$

satisfy "$I_{1,[out]} = \left(I_{1,[in]}\right)^{O_{1,[out]}}$."

However, most of the time, we will not be so formal, and would instead present the claim as:

Claim 2.2.2. (Example) If we let $a \in \mathbb{R}$, if $I_{1,[in]} = a$, then $I_{1,[out]} = a^n$ where $O_{1,[out]} = n$.

How to prove claims such as above is described in detail in [26, pp. 13-18]. It is simply a matter of going through each steps of the algorithm, and show that

provided the value of the variables coming into the step satisfy some property, then the values coming out of the step will follow another property. By cleverly choosing the properties between steps, we can link the condition on $\left(I_{1,[in]}, ..., I_{n,[in]}\right)$ to the one on $\left(I_{1,[out]}, ..., I_{n,[out]}, O_{1,[out]}, ..., O_{m,[out]}\right)$. Choosing the right properties is usually the hard part.

2.3 Sums in algorithms

To remove any ambiguity, within the following algorithms, sums such as

$$\sum_{a<b} E\left(a, b\right)$$

will always sum over (potentially multiple) values of a, but only one value of b. In other words, the first variable below the sum is the one summed over, and the second is a fixed constraint. If we wish to sum over all pairs a, b such that $a < b$, we will write $\sum_b \sum_{a<b} E\left(a, b\right)$.

We interpret such sums as a sequential series of additions. It is true that summation is a *reduction*[2] with a binary operator (addition) which is both commutative and associative, and that therefore it could be executed in parallel [22]. In the analysis of the performance of the algorithms, we assume that we do not reduce sums in parallel, unless otherwise specified. However, if after having implemented one of the following algorithms and *profiling*[3], the implementer determines that a significant portion of time is spent in the summations and that there are still processors available, then it is advised to try a parallel reduction instead.

[2]This term is used [22] to refer to the application of a binary operation $\oplus : Z \times Z \to Z$ to a list of elements $(z_i)_{i=1}^N \in Z$, written $\oplus_{i=1}^N z_i$.

[3]Profiling a program consists in executing it and counting the number of CPU cycles spent at each step. By seeing where most of the time is spent, we can focus the attention where the overall performance can be most affected.

2.4 Asymptotic Notation

In what follows, we will denote $\{x \in \mathbb{R} \,|x > 0\}$ by $\mathbb{R}_+^\star$.

Definition 2.4.1. (Asymptotic Notation, see [26, Vol 1, pp. 107-111]). To every function $f : \mathbb{N} \to \mathbb{R}_+^\star$, we associate several sets of functions:

- For any constant $C \geq 0$, we define

$$\Theta_C(f) = \left\{ g : \mathbb{N} \to \mathbb{R}_+^\star \,\middle|\, \lim_{m\to\infty} \frac{g(m)}{f(m)} \text{ exists and equals } C \right\}$$

  the set of functions approximately equal to $Cf(m)$ in the limit.

- Define $o(f) = \Theta_0(f)$, the set of functions *strictly dominated by* f.

- Define $\Theta(f) = \cup_{C>0}\Theta_C(f)$, the set of functions asymptotically equal to f, up to some positive constant.

- Define $\mathcal{O}(f) = \cup_{C\geq 0}\Theta_C(f) = o(f) \cup \Theta(f)$, the set of functions *dominated by* f.

- Define $\Omega(f) = \left\{ g : \mathbb{N} \to \mathbb{R}_+^\star \,\middle|\, \lim_{m\to\infty} \frac{f(m)}{g(m)} = 0 \right\}$, the set of functions *dominating* f.

Remark 2.4.2. Let $h : \left(\mathbb{R}_+^\star\right)^k \to \mathbb{R}_+^\star$, and $A_1, A_2, ..., A_k : \mathbb{N} \to \mathbb{R}_+^\star$. We shall denote

$$\left\{ f : \mathbb{N} \to \mathbb{R}_+^\star \,|\exists_{f_1\in A_1, f_2\in A_2, ..., f_k\in A_k} \forall_{m\in\mathbb{N}}. f(m) = h(f_1(m), f_2(m), ..., f_k(m)) \right\}$$

by $h(A_1, ..., A_k)$.

Remark 2.4.3. Also note that if we define a relation between functions so that $g \equiv f$ if and only if $g \in \Theta(f)$, this is an equivalence relation. Expressions written with Θ can be manipulated algebraically, so that the following hold:

1. $\Theta(f) + \Theta(g) = \Theta(f + g)$. Furthemore, if $g \in \mathcal{O}(f)$, then $\Theta(f) + \Theta(g) = \Theta(f)$.
2. $\frac{\Theta(f)}{\Theta(g)} = \Theta\left(\frac{f}{g}\right)$.

3. $\Theta(f)\Theta(g) = \Theta(fg)$

The first can be generalized a little to $\sum_{i=1}^{k} \Theta(f_i) = \Theta\left(\sum_{i=1}^{k} f_i\right)$. Furthermore, if $\forall_i . f_i \in \mathcal{O}(f_j)$ for some f_j, then $\sum_{i=1}^{k} \Theta(f_i) = \Theta(f_j)$ as long as k is independent of m.

2.5 Parallelism

In this section, we do not give a complete treatment of parallelism as it would distract from rather than help support the core of this thesis. Between simply hoping that the reader's intuition coheres with our own and mathematically constructing the machine architecture we have in mind, we will try for a compromise. We will assume that the idea of a *serial* computer is clear to the reader (see [26, Vol 1, pp. 124-164] for an example of a serial computer described precisely), and we shall express the parallel model in terms of the serial model. There are many different models for parallel architectures. Ultimately, we are interested in the actual architecture of a modern General Purpose Graphical Processing Unit, such as the GTX960, by NVIDIA. However, for simplicity of presentation, we shall describe a rudimentary model for a parallel computer, and discuss the parallel aspects of the performance of our algorithms in terms of this model. For more on the architecture of parallel computers, see [39, 12].

In the context of a parallel computer, we will refer to individual serial sub-computers as *processors*. Our parallel computer will have many processors, each executing instructions like a serial computer would.

Our parallel architecture has one "Control processor," and potentially many "Slave processors."

Once we have fixed an algorithm together with the initial values of its inputs, we get a (potentially infinite) sequence of *batches* $(B_i)_{i\in\mathbb{N}}$, each of which will be a (finite) list of *tasks* T_{ij}, *so that* $B_i = (T_{ij})_{j=1}^{N_i}$ for $i \in \mathbb{N}$. Each task is just a (potentially infinite) sequence of elementary operations. As an example, if at the i-th step, an algorithm states to "$[\![\forall_{j\in\{1,\ldots,n\}}]\!]$, set $X_j \leftarrow X_j^2$," then this would be a single batch B_i

composed of n tasks T_{ij}, each of which consisting of the squaring operation followed by the assignment operation.

Alternatively, we can consider the set of all possible tasks Task to be the set of all possible serial algorithms (see [26, Vol 1, pp. 124-164]). Then, Batch $= \cup_{n\in\mathbb{N}} (\text{Task})^n$ gives the set of all possible batches, and finally, Exec $= \text{Batch}^{\mathbb{N}}$ gives the set of possible executions of parallel algorithms. Then, algorithms given in the form of Algorithm X are simply a short-hand description of how to construct the execution once we have been given the input. Also notice that algorithms which terminate will give rise to executions whose batches are eventually empty.

To make our parallel architecture precise, we give the algorithm that the Control processor will follow to make the parallel computer execute $B \in$ Exec.

But first, for every batch, we define the function

$$\text{Count} : \text{Batch} \rightarrow \mathbb{N}$$

such that if $b \in \text{Task}^N$, Count $(b) = N$. Furthermore, we shall use $b(j)$ to denote $\Pi_j b$, the j-th task in b.

Algorithm C (*Control Loop*). This is the program that the Control Processor will run to operate the parallel computer. We invoke it as Control(B, p).

C1. [Start program] Set $i \leftarrow 1$. (i will be the Batch index.)

C2. [Start Batch.] Set $\kappa \leftarrow$ Count (B_i), and $c \leftarrow 1$. (κ will be the number of tasks, c will be the first task yet to be started.)

C3. [No more tasks?] If $\kappa = 0$, terminate.

C4. [Remaining tasks $\geq$ processors?] If $p \leq \kappa$, then set $s \leftarrow p$. (s will be the number of tasks to start)

C5. [Remaining tasks $<$ processors?] If $p > \kappa$, then set $s \leftarrow \kappa$.

C6. [Start s tasks and then wait] Start processors 1 through s with tasks $B_i(c)$ through $B_i(c+s)$. Wait for processors 1 through s to complete their tasks, while counting the time steps spent by the Slave processors.

C7. [Update] Set $\kappa \leftarrow \kappa - s$, and $c \leftarrow c + s$. (We reduce the number of remaining

tasks, and increase the index of the first task yet to be started.)

C8. [More tasks?] If $\kappa > 0$, go back to C4.

C9. [Move to next Batch] If $\kappa = 0$, set $i \leftarrow i + 1$, and go back to C2. ∎

Note that step C6 implies that the slowest processor determines how much time a task takes. Furthermore, if two processors start their tasks simultaneously, we know that both will have returned by the time another task is started on either one. This is unrealistic. We also cannot have recursive parallelism where a Slave processor would start tasks on some other Slave processors.

In case the notion of time in a parallel computer is confusing, we define elementary operations (comparison of integers, comparison of real values; modular, real, and complex arithmetic; going to a new step; assigning a value to a variable; computing the complex exponential of a given angle; computing the phase of a complex number, alternatively, computing the arc tangent of a point on the plane, etc.), and stipulate that *every processor currently working on a task will execute exactly one elementary operation at each time step.* The Control processor counts the time spent waiting for the started tasks to complete on step C6. For any given waiting period, if the working processor which takes the most time steps to complete takes $t_{max} \in \mathbb{N}$ steps, then the Control processor will experience t_{max} time steps waiting.

Definition 2.5.1. (Parallel performance metrics). In this way, we define a function

$$\text{ControlTime} : \text{Exec} \times (\mathbb{N} \setminus \{0\}) \to \mathbb{N}$$

which takes an execution, as a sequence of batches, and a number of slave processors, and returns the number of time steps between the moment the control processor begins the execution and the moment it terminates.

- The *Work* is defined as $\Upsilon = \text{ControlTime}\,(B, 1)$.
- The *Span* is $\Upsilon_\infty = \lim_{p\to\infty} \text{ControlTime}\,(B, p)$.

- The *Speedup* is defined in terms of p the number of processors as

$$S_p = \frac{\text{ControlTime}\,(B,1)}{\text{ControlTime}\,(B,p)}$$

.

- If $S_p \in \Theta\,(p)$, we say that the speedup for p processors is *linear*.
- The *Parallelism* is defined as $S_\infty = \lim_{p\to\infty} S_p$.
- With this, we can say that $S_\infty = \frac{\Upsilon}{\Upsilon_\infty}$, or $\Upsilon_\infty = \frac{\Upsilon}{S_\infty}$ (the span is the work over the parallelism).
- If $S_{S_\infty} \in \Theta\,(S_\infty)$, we say that the parallelism is *linearly accessible*. In other words, up to a constant, the maximum speedup can be reached asymptotically with a number of processors such that the speedup is linear.
- The *efficiency* is defined as $\frac{S_p}{p}$.

With this, we can say that the parallelism is linearly accessible if there is p such that $S_p \in \Theta\,(S_\infty)$ and $\frac{S_p}{p} \in \Theta\,(1)$.

Since by construction, for a given algorithm and a fixed input, we get an element of Exec, we can also consider the function which gives the control time in terms of the input and of the number of processors, for a fixed algorithm A. By extension, the work, span, etc. can be viewed as functions of the algorithm, the input, and the number of available processors.

Remark 2.5.2. These concepts, aside from "linearly accessible parallelism," are standard terminology for the analysis of the performance of parallel algorithms, for instance, see [11, pp. 779-784]

Remark 2.5.3. Certain combinations of these quantities are especially convenient, since knowing them for individual batches $B_1, ..., B_k$ is sometimes enough to determine them for the sequence $B = (B_i)_{i=1}^k$. For instance, we can choose (work, span, parallelism is linearly accessible) or we can choose (work, parallelism, parallelism is

linearly accessible). These two are equivalent, since $S_\infty = \frac{\Upsilon}{\Upsilon_\infty}$. We formalize this idea in the following lemma.

Lemma 2.5.4. *(Convenience of work, span, and linear accessibility to decompose performance analysis). Let $B_1, B_2, ..., B_k$ be k batches, and $B = (B_i)_{i=1}^k$. Also, let the works be $\Upsilon(B_i)$ and the spans be $\Upsilon_\infty(B_i)$. Finally, let all the B_is have a linearly accessible parallelism ($\forall_i . S_{S_\infty(B_i)} \in \Theta(S_\infty(B_i))$.*

Then, the work of the compounded sequence of batches is the sum of the works $\Upsilon(B) = \sum_{i=1}^k \Upsilon(B_i)$, the span is the sum of the spans $\Upsilon_\infty(B) = \sum_{i=1}^k \Upsilon_\infty(B_i)$, and the question of whether the parallelism is linearly accessible can be settled by the following:

If we let $p = S_\infty(B)$ be the parallelism of the compounded sequence of batches, then the speedup for p processors is given in terms of the individual pieces by

$$S_p(B) = \frac{\Upsilon(B)}{\sum_{i=1}^k \frac{\Upsilon(B_i)}{S_p(B_i)}}$$

where the individual speedups are

$$S_p(B_i) \in \Theta(\min(p, S_\infty(B_i)))$$

Then, it has linear accessibility if $S_p(B) \in \Theta(p)$.

The proof is an elementary application of the definitions, together with Remark 2.4.3, except perhaps for the fact that $S_p \in \Theta(\min(p, S_\infty))$ for every p when the parallelism is linearly accessible. To see this, first we see that $S_p \leq S_\infty$ by considering Algorithm C. Furthermore, we have that if $\frac{S_p}{p} \in \Theta(1)$, then $\frac{S_{p'}}{p'} \in \Theta(1)$ for $1 \leq p' \leq p$. This is true because, p' processors can do what p processors did before in $\Theta\left(\left\lceil \frac{p}{p'} \right\rceil\right) = \Theta\left(\frac{p}{p'}\right)$ times as many steps.

Chapter 3

An Introduction to Optimization

3.1 The Optimization Problem

In this section, unless otherwise specified, X will be a set, $f : X \to \mathbb{R}$ will be bounded, and $y \in \mathbb{R}$.

Example 3.1.1. Fixing m, consider $\mathcal{M}_m(\mathbb{R})$ the set of real-valued $(m \times m)$ matrices and $S = \left\{M \in \mathcal{M}_m(\mathbb{R}) \,\middle|\, \nexists_{A \in \mathcal{M}_m(\mathbb{R})}. AM = MA = \mathbb{I}\right\}$ the subset of singular matrices. We know that $\det(M) = 0 \iff M \in S$. Furthermore, we know that $\min(f(\mathcal{M}_m(\mathbb{R}))) = 0$ if $f : \mathcal{M}_m(\mathbb{R}) \to \mathbb{R}$ is given by $f(M) = |\det(M)|$. Therefore, we have that $S = f^{-1}(\min(f(\mathcal{M}_m(\mathbb{R}))))$ and $S = f^{-1}(0)$. In some sense, $\mathcal{M}_m(\mathbb{R})$, f, and 0 characterize S.

In general, we might have X, $f : X \to \mathbb{R}$ and $y \in \mathbb{R}$, yet only know that a subset $Y \subset X$ is given by $f^{-1}(\min(f(X)))$ *if* $Y \neq \emptyset$. Furthermore, we might know that $f(Y) = \{y\}$ (if $Y \neq \emptyset$). In other words, we do not know if $f(Y) = \{\min f(X)\}$, but when given $\min f(X)$, we can check against y to see if $Y \neq \emptyset$. Also, without extra hypotheses, we do not know if $\min f(X)$ will be defined, so we use the infimum instead.

We can get an expression which works in all cases, as in the following definition.

Definition 3.1.2. Given X, f, y as above, and $Y \subset X$, we say that (X, f, y) *characterizes* Y if

$$Y = f^{-1}\left(\inf\left(f\left(X\right)\right)\right) \cap f^{-1}\left(y\right)$$

In which case, we write $\text{Optimize}_{X,f,y} = Y$.

Lemma 3.1.3. *For all $Y \subset X$, there exists f and y as above such that (X, f, y) characterizes Y.*

Proof. For a given $Y \subset X$, consider $(X, -\mathbf{1}_Y, -1)$, where $\mathbf{1}_Y : X \to \mathbb{R}$ is the indicator function of Y, with

$$\mathbf{1}_Y\left(x\right) = \begin{cases} 1 & x \in Y \\ 0 & x \notin Y \end{cases}$$

Then, we consider two cases. First, if $Y = \emptyset$, then $\inf\left(-\mathbf{1}_Y\left(X\right)\right) = 0$ and so the preimage of $\inf\left(-\mathbf{1}_Y\left(X\right)\right)$ will be X, but the preimage of -1 will be $\emptyset$. Therefore,

$$\text{Optimize}_{X,-\mathbf{1}_Y,-1} = X \cap \emptyset = \emptyset = Y$$

Second, if $Y \neq \emptyset$, then $\inf\left(-\mathbf{1}_Y\left(X\right)\right) = -1$, with preimage Y, and so

$$\text{Optimize}_{X,-\mathbf{1}_Y,-1} = Y \cap Y = Y$$

□

The optimization problem, then, is simply to, given X, f, y, find $x \in \text{Optimize}_{X,f,y}$. Note that this implies that the optimization problem is impossible to solve when $\text{Optimize}_{X,f,y} = \emptyset$. Lemma 3.1.3 tells us that any problem of the form "If $Y \neq \emptyset$, then find $x \in Y$" can be rephrased as an optimization problem.

Remark 3.1.4. In general, for $Y \subset X$, the set $\left\{(f, y) \middle| \text{Optimize}_{X,f,y} = Y\right\}$ contains more than one element. Furthermore, depending on which (f, y) we choose to characterize Y, solving the optimization problem may be easier or harder to solve.

Example 3.1.5. Set $X = \mathbb{Z}$, $Y = \{0\}$, and consider $f_1, f_2 \in \mathbb{R}^X$ bounded and

$y_1, y_2 \in \mathbb{R}$ given by $f_1(x) = \begin{cases} 0 & x = 0 \\ 1 & x \neq 0 \end{cases}$, $f_2(x) = |x|$, $y_1 = y_2 = 0$. We have that $Y = \text{Optimize}_{X,f_1,y_1} = \text{Optimize}_{X,f_2,y_2}$. However, the simple iterative function $I_f : X \to X$ given by

$$I_f(x) = \begin{cases} x-1 & f(x-1) \leq f(x+1) \text{ and } f(x-1) \leq f(x) \\ x & f(x) > f(x-1) \text{ and } f(x) > f(x+1) \\ x+1 & \text{Otherwise} \end{cases}$$

is such that the sequence $\left(x, I_f(x), I_f(I_f(x)), ..., I_f^i(x), ...\right)$ converges to 0 for every $x \in X$ if $f = f_2$, and yet this sequence never converges if $x < 0$ and $f = f_1$.

Sometimes, we might be interested in the preimage of the minimum of a given function, without knowing what the value of the minimum is.

Definition 3.1.6. Given X, f, as above, and $Y \subset X$, we say that (X, f) *characterizes* Y if

$$Y = \cup_{y \in \mathbb{R}} \text{Optimize}_{X,f,y}$$

In which case, we write $\text{Optimize}_{X,f} = Y$.

Remark 3.1.7. The subset $\text{Optimize}_{X,f}$ can sometimes be empty (e.g. $X = \mathbb{R} \setminus \{0\}$, $f(x) = |x|$). However, if $|X| < \infty$ and $X \neq \emptyset$, then $\text{Optimize}_{X,f} \neq \emptyset$.

We can also extend these notions to a relaxed version. We recall that, within a metric space (Z, d), if $\varepsilon > 0$ and $z \in Z$, then $\mathcal{B}_\varepsilon(z) = \{z' \in Z \,|\, d(z, z') < \varepsilon\}$ is the open ball around z.

Definition 3.1.8. Given X, f, y as above, $\varepsilon > 0$ and $Y \subset X$, we say that (X, f, y) *ε-characterizes* Y if

$$Y = f^{-1}\left(\mathcal{B}_\varepsilon\left(\inf\left(f(X)\right)\right)\right) \cap f^{-1}\left(\mathcal{B}_\varepsilon(y)\right)$$

In which case, we write $\text{Optimize}_{X,f,y}[\varepsilon] = Y$. Similarly, we say that (X, f) *ε-characterizes* Y if

$$Y = \cup_{y \in \mathbb{R}} \text{Optimize}_{X,f,y}[\varepsilon]$$

In which case, we write $\text{Optimize}_{X,f}[\varepsilon] = Y$.

We will sometimes refer to the task of finding $x \in \text{Optimize}_{X,f,y}[\varepsilon]$ as the ε-relaxed optimization problem. We see that for ε sufficiently large, the ε-relaxed optimization problem will be easier to solve than the optimization problem itself (unless $X = \emptyset$).

Since $\cap_{\varepsilon>0}\mathcal{B}_\varepsilon(x) = \{x\}$, we have that

$$\text{Optimize}_{X,f,y} = \cap_{\varepsilon>0}\text{Optimize}_{X,f,y}[\varepsilon]$$

and also

$$\text{Optimize}_{X,f} = \cap_{\varepsilon>0}\text{Optimize}_{X,f}[\varepsilon]$$

In general, for $\varepsilon > 0$,

$$\text{Optimize}_{X,f,y} \neq \text{Optimize}_{X,f,y}[\varepsilon]$$

However, there will be cases where $\text{Optimize}_{X,f,y}[\varepsilon]$ is "a good approximation of" $\text{Optimize}_{X,f,y}$.

Definition 3.1.9. Let $\varepsilon, \delta > 0$, (X, d) a metric space, with f, y as above. We say that (X, f, y) is (ε, δ)-*faithful*[1] if

$$\text{Optimize}_{X,f,y}[\varepsilon] \subset \cup_{z\in\text{Optimize}_{X,f,y}}\mathcal{B}_\delta(z)$$

Similarly, we say that (X, f) is (ε, δ)-*faithful* if

$$\text{Optimize}_{X,f}[\varepsilon] \subset \cup_{z\in\text{Optimize}_{X,f}}\mathcal{B}_\delta(z)$$

Definition 3.1.10. Let $\varepsilon, \delta > 0$, (X, d) a metric space, with f, y as above, and $q \in X$. We say that q is an (ε, δ)-Quasi-Min[2] of (X, f, y) if $q \in \text{Optimize}_{X,f,y}[\varepsilon]$ and yet $q \notin \cup_{z\in\text{Optimize}_{X,f,y}}\mathcal{B}_\delta(z)$.

[1] This is not, as far as we know, standard terminology. We chose "faithful" to invoke "a good approximation." We can see immediately that, if (X, f, y) is (ε, δ)-faithful, then any point $x \in \text{Optimize}_{X,f,y}[\varepsilon]$ will be at distance less than δ of at least one point in $\text{Optimize}_{X,f,y}$.

[2] This is, as far as we know, not a standard concept in the literature. Also, it has nothing to do with other common usages of the terms "Quasi minima," as in [19].

Similarly, q is an (ε, δ)-Quasi-Min of (X, f) if $q \in \text{Optimize}_{X,f}[\varepsilon]$ and yet $q \notin \cup_{z \in \text{Optimize}_{X,f}} \mathcal{B}_\delta(z)$.

This gives an obvious, but still interesting property.

Proposition 3.1.11. *Keeping the above notation, there exist some (ε, δ)-Quasi-Min of (X, f, y), if and only if (X, f, y) is not (ε, δ)-faithful.*

Similarly, there exist some (ε, δ)-Quasi-Min of (X, f), if and only if (X, f) is not (ε, δ)-faithful.

Proof. Both of these cases follow directly from the fact that, if $A, B \subset X$, then $A \subset B$ if and only if $A \setminus B = \emptyset$. Indeed, we can take $A = \text{Optimize}_{X,f,y}[\varepsilon]$ and $B = \cup_{z \in \text{Optimize}_{X,f,y}} \mathcal{B}_\delta(z)$ to prove the first case, and similarly for the second case. □

Example 3.1.12. We set $X = \mathbb{R}$, with the euclidean distance, $f(x) = |x|$, $y = 0$, $\varepsilon = 1, \delta = \frac{1}{2}$. Then, the set of all (ε, δ)-Quasi-Mins of (X, f, y) is

$$(-1, 1) \setminus \left(-\frac{1}{2}, \frac{1}{2}\right) = \left(-1, -\frac{1}{2}\right] \cup \left[\frac{1}{2}, 1\right)$$

Remark 3.1.13. In practice, we can imagine solving the optimization problem, for a given (X, f, y) by first introducing $\varepsilon > 0$, solving the ε-relaxed optimization problem, and then using the solution $x \in \text{Optimize}_{X,f,y}[\varepsilon]$ to find $x' \in \text{Optimize}_{X,f,y}$. Let us imagine a case where this becomes harder and harder to do as $d\left(x, \text{Optimize}_{X,f,y}\right)$ increases. Then, we can well imagine cases where the presence of (ε, δ)-Quasi-Mins, when δ is large enough, could make this strategy fail. In Theorem 6.3.6, we give an example of such a point $x \in \text{Optimize}_{X,f,y}[\varepsilon]$ in a context where $\text{Optimize}_{X,f,y} = \mathcal{H}_m$ is the set of Hadamard matrices.

Note however, that if we have f_1, f_2, y_1, y_2 such that

$$\text{Optimize}_{X,f_1,y_1} = \text{Optimize}_{X,f_2,y_2}$$

and that the (ε, δ)-Quasi-Mins of (X, f_1, y_1) and those of (X, f_2, y_2) are problematic (as above), then it might be that the sets of problematic points for both cases are disjoint from each other. In such a scenario, there might be tricks to be played with

f_1 and f_2 during the optimization to avoid the problematic points altogether (e.g. using $f_1 + f_2$ instead, or alternating between using f_1 and f_2).

Now we turn to cases where the two sets are not disjoint.

Definition 3.1.14. Let $\varepsilon_1, ..., \varepsilon_n, \delta > 0$, (X, d) a metric space, and $f_1, ..., f_n \in \mathbb{R}^X$ bounded functions, and $y_1, ..., y_n \in \mathbb{R}$ such that for all i, j, $\text{Optimize}_{X,f_i,y_i} = \text{Optimize}_{X,f_j,y_i}$, and finally $q \in X$. We say that q is an $(\varepsilon_1, ..., \varepsilon_n, \delta)$-Co-Quasi-Min of $(X, (f_i, y_i)_{i=1}^n)$ if, for every i, it is an (ε_i, δ)-Quasi-Min of (X, f_i, y_i).

The presence of (ε, δ)-Quasi-Mins obviously is a potential nuisance for optimization. Another one is the presence of δ-Local-Mins. These can be defined with respect to any metric structure on (X, d), or with respect to some digraph structure, or even some random digraph structure over X. Since not much is gained in the complication, we shall only define it for a metric space structure.

Definition 3.1.15. Let (X, d) a metric space, $f \in \mathbb{R}^X$ a bounded function, and $\delta > 0$. We say that $l \in X$ is a δ-Local-Min if

$$l \in f^{-1}\left(\inf f\left(\mathcal{B}_\delta\left(l\right)\right)\right)$$

If l is a δ-Local-Min such that $f\left(l\right) < f\left(x\right)$ for every $x \neq l$ in $\mathcal{B}_\delta\left(l\right)$, we say that l is a *strict* δ-Local-Min.

Example 3.1.16. Set $X = \mathbb{R}$ and d the usual euclidean distance, and set

$$f\left(x\right) = \begin{cases} 1 & x \neq 0 \\ 0 & x = 0 \end{cases}$$

with $y = 0$. Then, for a fixed $\delta > 0$, the set of all δ-Local-Mins is

$$\left(\mathbb{R} \setminus \left[-\delta, \delta\right]\right) \cup \left\{0\right\}$$

Furthermore, the set of strict δ-Local-Mins is $\{0\}$.

Remark 3.1.17. Comparing the Local-Mins to the Quasi-Mins, we see that there might be points $x \in X$ which are at once a δ_1-Local-Min and a (ε, δ_2)-Quasi-Min for

some values of $\delta_1, \delta_2, \varepsilon$. However, it is rare that being a δ_1-Local-Min implies being a (ε, δ_2)-Quasi-Min and vice-versa.

Remark 3.1.18. The standard definitions for the optimization problem, or the approximation problem (see for instance [4]) do a few things that we have not done

- They constrain the problem, and classify various sub-problems.
- They envisage solutions to this problem, and set themselves up to assess the performance of such solutions.
- They include in the definitions details meant to capture some classes of problems such as graph-theoretic problems, for combinatorial problems or problems over $\mathbb{R}^n$ for continuous problems.

It is of course a valid point of view, which might perhaps be taken as a future inquiry. In fact, from a theoretical standpoint (the theory of computation), establishing bounds and guarantees on the performance of potential optimization techniques would be a central concern. However, from another theoretical standpoint (the theory of Hadamard matrices), these bounds and guarantees would neither be necessary nor would they be sufficient to prove the existence of Hadamard matrices or (ε, δ)-Quasi-Hadamard matrices. Yet, knowing that a tuple (X, f, f_0) characterizes the Hadamard matrices is a fact *about Hadamard matrices.*

We therefore have two points of view when it comes to discussing optimization and Hadamard matrices. The first considers the optimization problem, and attempts to determine, for a given class of functions, the work required by the best algorithm which always succeeds, for instance. The second point of view is interested in finding tuples (X, f, f_0) which characterize the Hadamard matrices, and for such tuples, discuss the presence of (ε, δ)-Quasi-Mins. Of course, we would still want to find Hadamard matrices, and so the optimization problem is relevant, but the definitions are aimed at understanding Hadamard matrices better.

Remark 3.1.19. Since we have taken the second point of view as central in this thesis, our definitions:

- Constrains the problem to a lesser extent; such that it could easily be considered for any tuple (X, f, f_0) with $f : X \to \mathbb{R}$ and $f_0 \in \mathbb{R}$, or even just (X, Y) with $Y \subset X$, for any set X, no matter how big, for instance.
- Give no framework or specification of the kinds of algorithms which could be used to solve the problem. We merely give a way to judge whether the problem was solved.
- Presents only the general notions, stripped of details as much as possible.
- Asks questions about (X, f, f_0), such as the presence of Quasi-Mins or Local-Mins which are in principle independent of any algorithm to solve the problem. Yet in practice such algorithms might be used to answer such questions (even if they cannot solve the optimization problem, or even the relaxed optimization problem).

3.2 Basic Optimization Methods

We give a somewhat systematic description of various optimization methods. With perhaps the exception of the Multi-Metropolis methods, all optimization methods presented here are found in the literature [44, 41, 10, 23]. However, by their nature, they come from disparate contexts, so we only give the basic techniques directly related to those we have ended up using in the following chapters.

For each optimization method, there are many variations, many different algorithms which have either been derived from a common, simpler algorithm, or which have been grouped together in the literature because of similarities or possible analogies.

We will only give one algorithm per optimization method, or family of similar algorithms, then we explore the commonalities between such algorithms, we devise our own algorithms, and we study the parallelism of these algorithms.

Finally, we present some gradient methods, which we have not used directly, but for which we have devised efficient algorithms, to attack the problem of finding complex Hadamard matrices.

Algorithm EX (*Exhaustive Search*). Given a finite set X and a function $f \in \mathbb{R}^X$, we will find $\min f(X)$ as well as an element of $f^{-1}(\min f(X))$. We fix an order on X, and give it as the list of its elements $(x_i)_{i=1}^{|X|}$. X and f will not change during the algorithm, and after termination, we will have $\gamma_{min} = \min f(X)$ and $f(x_{min}) = \gamma_{min}$. We would invoke it as $x_{min}, \gamma_{min} \leftarrow \text{Exhaustive}(X, f)$.

EX1.[Initialize.] Set $i \leftarrow 1$, $x \leftarrow x_i$, $\gamma \leftarrow f(x)$, $x_{min} \leftarrow x$ and $\gamma_{min} \leftarrow \gamma$. (After this, $\gamma_{min} = \min_{n \in \{1,\ldots,i\}} f(x_i)$ and $f(x_{min}) = \gamma_{min}$.)

EX2.[Go through all of X.] Set $i \leftarrow i + 1$. If $i > |X|$, terminate. (After this, $\gamma_{min} = \min_{n \in \{1,\ldots,i-1\}} f(x_i)$ and $f(x_{min}) = \gamma_{min}$.)

EX3.[Get current.] Set $x \leftarrow x_i$, and $\gamma \leftarrow f(x)$.

EX4.[New min?] If $\gamma_{min} > \gamma$, set $\gamma_{min} \leftarrow \gamma$, and $x_{min} \leftarrow x$. (After this, $\gamma_{min} = \min_{n \in \{1,\ldots,i\}} f(x_i)$ and $f(x_{min}) = \gamma_{min}$.)

EX5.[Loop.] Go back to EX2. ∎

Remark 3.2.1. Algorithm EX is surely one of the simplest optimization methods, and it is guaranteed that if it terminates, it will have solved the optimization problem. However, if we fix the resources (e.g. time), then we will always be able to find X with $|X|$ too large to reach termination within the allowed ressources. Since not all elements of X would have been tried, there would be no guarantees that some other method couldn't have done much better. However, we will combine algorithm EX with every other iterative algorithm, so that, we may remember the best element visited during execution.

Algorithm RS (*Randomized Search*). Let X be a set, and let Sampler be a random variable in X^Ω distributed according to a distribution $\mathcal{D}$. Given Sampler, a function $f \in \mathbb{R}^X$, and a positive integer $n \in \mathbb{N} \setminus \{0\}$, we repetitively ($n$ times) sample from Sampler, thus we visit a list $(x_i)_{i=1}^n$ of not necessary distinct elements of X. After termination, we will have $\gamma_{min} = \min_{i \in \{1,\ldots,n\}} f(x_i)$ and $f(x_{min}) = \gamma_{min}$, such that $x_{min} = x_i$ for some i. We would invoke it as $x_{min}, \gamma_{min} \leftarrow \text{Randomized}(\text{Sampler}, f, n)$.

RS1. [Initialize.] Set $i \leftarrow 1, x \leftarrow (\!|\text{Sampler}|\!)$ $\gamma \leftarrow f(x)$, $x_{min} \leftarrow x$, and $\gamma_{min} \leftarrow \gamma$.

RS2. [Go through n samples.] Set $i \leftarrow i + 1$. If $i > n$, terminate.

RS3. [Get sample.] Set $x \leftarrow$ (|Sampler|), and $\gamma \leftarrow f(x)$.

RS4. [New min?] If $\gamma_{min} > \gamma$, set $\gamma_{min} \leftarrow \gamma$, and $x_{min} \leftarrow x$.

RS5. [Loop.] Go back to RS2. ∎

Remark 3.2.2. Simulating randomness on a computer is not simple, and so we shall not get into the details for now, but it can be done. (see [28, Vol 2].)

Remark 3.2.3. The probability of not having found an element of $Y \subset X$ with the randomized search in n steps is given by $(P[\text{Sampler} \notin Y])^n$. An important special case is when $|X| < \infty$, but very big, and the distribution is uniform $\mathcal{D} = \sum_{x \in X} \frac{1}{|X|} \delta_x$. In this case, $P[\text{Sampler} \notin Y] = \frac{|X \setminus Y|}{|X|} = 1 - \frac{|Y|}{|X|}$. As we can see, $|X|$ in itself doesn't matter. Rather, if such a method fails with a high probability, it is because $\frac{|Y|}{|X|}$ is too small. This is well-known in the sampling community.

Therefore, success depends on the size of X (and the order in which we list the elements) for algorithm EX and on the density of $f^{-1}(\inf f(X))$ within X for algorithm RS.

If both of these fail us, we need to start navigating through X using f in some way, and therefore, predicting what happens is no longer so simple.

Algorithm HC (*Hill Climbing*[3]). Let X be a set, and $\mathcal{G} \in \left(\cup_{n \in \mathbb{N} \setminus \{0\}} X^n\right)^X$. Let $x_0 \in X$, and $f \in \mathbb{R}^X$. Given $\mathcal{G}, f, x_0, n$, we start at x_0, and we repeatedly (n times) get a list of "potential points" or "candidate points" and use Exhaustive Search to find the best one, which we visit, and then repeat. Over all, we will have visited a list $(x_i)_{i=1}^n$ of not necessary distinct elements of X. After termination, we will have $\gamma_{min} = \min_{i \in \{1,\ldots,n\}} f(x_i)$ and $f(x_{min}) = \gamma_{min}$, as well as $\gamma = f(x_n), x = x_n$, respectively the best and the last pair of value and point we will have visited. We would invoke it as $x_{min}, \gamma_{min}, x, \gamma \leftarrow \text{HillClimb}(\mathcal{G}, f, x_0, n)$.

HC1.[Initialize.] Set $i \leftarrow 1, x \leftarrow x_0$, $\gamma \leftarrow f(x)$, $x_{min} \leftarrow x$, and $\gamma_{min} \leftarrow \gamma$.

HC2.[Go through n steps.] Set $i \leftarrow i + 1$. If $i > n$, terminate.

[3] *Obviously, this terminology makes more sense when trying to maximize a function, but this term is so ubiquitous in the optimization literature, and the alternative is simply not as catchy, so we shall use Hill Climbing to refer to the process described here, in the context of minimizing a function, since the rest of the thesis is concerned with minimization.*

HC3.[Get the candidates.] Call $C \leftarrow \mathcal{G}(x)$.

HC4.[Select optimal candidates.] Call $x, \gamma \leftarrow \text{Exhaustive}(C, f)$.

HC5.[New min?] If $\gamma_{min} > \gamma$, set $\gamma_{min} \leftarrow \gamma$, and $x_{min} \leftarrow x$.

HC6.[Loop.] Go back to HC2. ∎

Let us recall that Π_i is the projection onto the i-th component of a tuple.

Remark 3.2.4. If (X, d) is a finite metric space, a special case is given by taking the candidates to be the closed ball of radius 1 around the current point, but given as an ordered set, so that

$$\{\Pi_i \mathcal{G}(x) \,|i \text{ is between 1 and the dimension of } \mathcal{G}(x)\} = \overline{\mathcal{B}}_1(x) = \{y \in X \,|d(x, y) \leq 1\}$$

Notice that we can still define the Hill Climbing without the notion of a metric space. Indeed, any $\mathcal{G} \in \left(\cup_{n \in \mathbb{N} \setminus \{0\}} X^n\right)^X$ gives a kind of *digraph* structure, where x is sent to all the points y for which $x \to y$ in the digraph (with multiplicity in general, though for our context there is no point, really).

This digraph structure in turn defines a kind of "directed distance,"

$$d_{Digraph}(x, z)$$

as "the number of arcs in the shortest oriented path from x to z." (See [29, Vol 4A, p.19].)

This directed distance could be used to define a kind of Local-Min specific to a given Hill Climbing algorithm.

Remark 3.2.5. Usually, the Hill Climbing will be given a random variable $\mathcal{X}_0 \in X^\Omega$ as input $x_{0,[in]}$, and will be restarted several times with different (independent) random initial points. This is done to mitigate the tendency of getting stuck in some Local-Mins. Indeed, we see that, for any $\delta > 0$, if l is a strict δ-Local-Min such that $\mathcal{G}(l) \subset \mathcal{B}_\delta(l)$, then if Algorithm HC ever reaches step HC3 with x having value l, then x would still have that value after step HC3, after HC4, after HC5, after HC6, and finally also after HC2. Therefore, the next time through the loop, Algorithm HC

would still have x with value l at step HC3. Hence, the algorithm could not visit any point other than l afterwards. Unless l is a solution to the optimization problem, this would be a failure state.

One way to avoid this problem, is to stop doing "locally optimal choices" all of the time. For some other ways, see [41].

Algorithm MH (*Metropolis-Hastings, see [41]*). Let X be a set, and $G \in \left(X^{\Omega}\right)^{X}$. Also let Unif a random variable with value in the interval $[0,1)$, distributed uniformly. Let $x_0 \in X$, $f \in \mathbb{R}^X$, $n \in \mathbb{N} \setminus \{0\}$, and $T > 0$. We start at x_0, and we repeatedly (n times) get a single random "candidate point" and perform a "thermodynamical transition" to get the next point which we visit. Over all, we will have visited a list $(x_i)_{i=1}^{n}$ of not necessary distinct elements of X. After termination, we will have $\gamma_{min} = \min_{i\in\{1,\ldots,n\}} f(x_i)$ and $f(x_{min}) = \gamma_{min}$, as well as $\gamma = f(x_n), x = x_n$, respectively the best and the last pair of value and point we will have visited. We would invoke it as $x_{min}, \gamma_{min}, x, \gamma \leftarrow \text{Metro}(G, f, x_0, n, T)$.

MH1.[Initialize.] Set $i \leftarrow 1, x \leftarrow x_0$, $\gamma \leftarrow f(x)$, $x_{min} \leftarrow x$, and $\gamma_{min} \leftarrow \gamma$.

MH2.[Go through n steps.] Set $i \leftarrow i + 1$. If $i > n$, terminate.

MH3.[Get the candidate.] Set $x_{cand} \leftarrow (\!|G(x)|\!)$, and $\gamma_{cand} \leftarrow f(x_{cand})$.

MH4.[Transition.] Set $u \leftarrow (\!|\text{Unif}|\!)$. If $u \leq \exp\left(-\frac{\gamma_{cand}-\gamma}{T}\right)$, set $x \leftarrow x_{cand}$ and $\gamma \leftarrow \gamma_{cand}$. (Transition to x_{cand} with probability $\min\left(1, \exp\left(-\frac{\gamma_{cand}-\gamma}{T}\right)\right)$, otherwise do not change x.)

MH5.[New min?] If $\gamma_{min} > \gamma$, set $\gamma_{min} \leftarrow \gamma$, and $x_{min} \leftarrow x$.

MH6.[Loop.] Go back to MH2. ∎

Remark 3.2.6. This procedure can be used to simulate thermodynamical processes, and compute the averages of certain variables. In such a case, a natural choice for G would be $G(x) = \mathcal{X} \in X^{\Omega}$ where $\mathcal{X}$ is distributed uniformly[4], and does not depend on x. This would be a terrible choice for the purposes of solving the optimization problem. In such a case, n steps of the Metropolis-Hastings algorithm would visit

[4]These thermodynamical processes usually occur over a set of points X for which a concept of uniform distribution is defined, such as when X is finite, or isomorphic to $\mathbb{R}^N$ for some N.

a subset of the points visited by the Randomized Search with the same distribution as $\mathcal{X}$. The reason why this might be confusing is that, after a sufficient number of steps, MH would visit points x_i distributed according to a probability density function proportional to $\rho(x) = e^{-\frac{f(x)}{T}}$,which, for a suitable T will have a very high probability of being an element of $f^{-1}(\inf(f(X)))$, if such a set isn't empty. This is true, and yet, each successive point visited depends on the previous ones, and the only way that MH would have to increase the probability that $x_i \in f^{-1}(\inf(f(X)))$ would be to not transition once such a point was reached. Since the candidate is simply taken at random from a distribution independent of the current value of x, the first occurrence of a global minimum cannot come any sooner than it would without this "thermodynamical transition."

The fact that MH cannot be better than Randomized Search would still hold for any G which does not depend on x. More precisely, if $G(x)$ is given by the distribution $\mathcal{D}$, then MH (with this G) cannot be better (in the above sense) than Randomized Search (for than same $\mathcal{D}$). Such a scheme is in fact suitable for estimating thermodynamical averages [20], but not to deal with optimization.

But by choosing G to depend on x, it is possible to get something similar to Hill Climbing, but with a mechanism to escape Local-Mins. And this is a manifestation of the complexity creeping in. We see that the performance of the optimization method will actually depend on many different aspects, and so even when it seems as though we can give some kind of guarantee, it turns out to be illusory, when we take a closer look.

Remark 3.2.7. This algorithm takes as input a parameter T, called the "thermal energy," or sometimes the "temperature." When T is very close to 0, the algorithm will behave much like the Hill Climbing, but when T is very large, the algorithm will take many non-optimal local decisions, purely out of randomness.

In theory, this is rather difficult to see, but in practice, the choice of temperature has incredible repercussions on performance. In fact this is probably the most crucial general advice with such "thermodynamical transition" algorithms: systematically experiment with temperatures over a wide range until a peak in performance is found. However, this technique of selecting the right temperature is somewhat orthogonal to

everything else, and so we shall not discuss it further.

3.3 Chains

Recalling the note about probabilistic notations from [47, p.11], in what follows, we say that $\sum_{x \in X} p_x \delta_x$ is a discrete distribution such that the probability of x is given by p_x.

During the execution of these algorithms we visit a sequence of elements of X, and every algorithm can be thought of as a specification of what that sequence will actually be:

- For Exhaustive Search, this sequence is simply all the elements of X enumerated in some order, $(x_i)_{i=1}^{|X|}$ such that $\{x_i, i \in \{1, ..., |X|\}\} = X$.
- For Randomized Search, this sequence is $(\mathcal{X}_i)_{i \in \mathbb{N}}$, with the $\mathcal{X}_i \in X^\Omega$ independent, identically distributed according to the distribution of $\text{Sampler}_{[in]}$.
- For Hill Climbing, this sequence is $(x_i)_{i=0}^{\infty}$ given by $x_0 = x_{0[in]}$ and

$$x_i = \tau\left(x_{i-1}\right)$$

  where, if we let $\left(c\left(i\right)\right)_{i=1,...,n} = \mathcal{G}\left(x\right)$, and $\epsilon_i = f\left(c\left(i\right)\right)$ for every i, by seeing $\epsilon : \{1, ..., n\} \to \mathbb{R}$, we get

$$\tau\left(x\right) \quad = \quad c\left(\min\left(\text{Optimize}_{\{1,...,n\},\epsilon}\right)\right)$$

- For Metropolis-Hastings, this sequence is $(\mathcal{X}_i)_{i \in \mathbb{N}}$ with the $\mathcal{X}_i \in X^\Omega$ given by $x_0 = \mathcal{X}_0$ and, for all $i \geq 1$,

$$\mathcal{X}_i \quad = \quad \tau\left(\mathcal{X}_{i-1}\right)$$

  where $\tau\left(x\right)$ is equal to either $G\left(x\right)$ or to x, and follows the discrete distribution $p\delta_{G(x)} + (1-p)\delta_x$ with $p = \min\left(1, \exp\left(-\frac{f(G(x))-f(x)}{T}\right)\right)$.

With the exception of Exhaustive Search, all of these schemes share a similar structure.

Definition 3.3.1. (Transition Procedure). Let X be a set, and X^{Ω} be the set of X-valued random variables.

1. We say that a function $\tau : X \to X^{\Omega}$ is a *transition procedure*[5].

2. Letting $\mathcal{D}$ a distribution on X, we say that a sequence $(\mathcal{X}_i)_{i\in\mathbb{N}}$ of random variables is an *induced Markov chain*[6] of τ starting at $\mathcal{X}_0$ if for every $i \geq 1$,

$$\mathcal{X}_i = \tau(\mathcal{X}_{i-1})$$

3. Furthermore if we let a function $E : X \to \mathbb{R}$ (the energy), a function $\mathcal{G} : X \to \cup_{n\in\mathbb{N}\setminus\{0\}} \left(X^{\Omega}\right)^n$ (generation procedure), and a function $\mathcal{S} : \cup_{n\in\mathbb{N}\setminus\{0\}} (\mathbb{R})^n \to (\mathbb{N}\setminus\{0\})^{\Omega}$ (selection procedure), we say that τ *has* energy E, generation procedure $\mathcal{G}$, and selection procedure $\mathcal{S}$ if τ is given by

$$\begin{aligned} \tau(x) &= g_{\mathcal{S}\left((\epsilon_i)_{i=1,\ldots,n}\right)} \\ \forall_{i=1,\ldots,n}.\epsilon_i &= E(g_i) \\ (g_i)_{i=1,\ldots,n} &= \mathcal{G}(x) \end{aligned}$$

With this terminology, all the sequences above except that of Exhaustive Search are the induced Markov chains of transition procedures which have energy f, and their generation and selection procedures given in Table 3.1.

With this, we see that the algorithms are simply an Exhaustive Search applied to the set of the first n values in the appropriate induced Markov chain.

[5] In general for Markov Chains, we have the notion of a transition probability matrix, or transition kernel. A transition procedure is just a more direct way, in our context, to describe the transition kernel. See [20] for more on Markov Chain Theory.

[6] Of course, this terminology comes from the fact that such a sequence is a Markov chain, namely, that the distribution of $\mathcal{X}_i$ may depend on $\mathcal{X}_{i-1}$, but given $\mathcal{X}_{i-1}$, the conditional distribution of $\mathcal{X}_i$ is independent of all $\mathcal{X}_j$ with $j < i-1$. For a treatment of Markov chains, see [20].

Method	Generation	Selection
Randomized Search	$x \mapsto [\![\text{Sampler}_{[in]}]\!]$	$(\epsilon_1) \mapsto 1$
Hill Climbing	$\mathcal{G}$	$(\epsilon_i)_{i=1}^n \mapsto \min\left(\text{Optimize}_{\{1,\ldots,n\},\epsilon}\right)$
Metropolis-Hastings	$x \mapsto (x, G(x))$	$(\epsilon_1, \epsilon_2) \mapsto (1-p)\delta_1 + p\delta_2$ $p = \min\left(1, \exp\left(-\frac{\epsilon_2 - \epsilon_1}{T}\right)\right)$

Viewing some basic optimization techniques as transition procedures given by various generation and selection procedures. The notation is that of algorithms RS, HC, and MH.

Table 3.1: Basic optimization techniques as transition procedures.

Remark 3.3.2. Looking at Table 3.1, one might well wonder why there aren't more possibilities? Would it be, for instance, possible to combine the generation from Hill Climbing with a selection similar to that for Metropolis-Hastings?

There are ways of doing this for the purposes of "sampling" some thermodynamical variables, and they are similar to what we propose, but they operate under the constraint of sampling from a definite distribution, whereas for us, the precise shape of the invariant measure for the Markov chain does not matter (indeed, even the existence of such a measure isn't necessary). For more details on the so-called "Multiple-try Metropolis-Hastings," and how it manages to preserve the "detailed balance" property, useful for sampling guarantees, see [31]. Since the idea is pretty straightforward, especially by looking at Table 3.1, it is perhaps not surprising that our scheme was rediscovered before knowing of this older, related scheme.

Because of the sheer number of possibilities, and lack of clear theoretical path to decide which optimization method was better, we resorted to the obvious approach of trying many methods that didn't seem obviously bad, and seeing which performed best.

But since there are so many parameters in flight, and some methods are random while some aren't, what exactly do we mean by "better" and "performed best?" A complete answer is hard to give, but we still give an imperfect "working definition" of performance.

Definition 3.3.3. Letting a set X, a subset $Y \subset X$, and $(\mathcal{X}_i)_{i\in\mathbb{N}}$, with $\mathcal{X}_i \in X^\Omega$, and finally $P(\{\mathcal{X}_j \,|j \leq n\} \cap Y \neq \emptyset)$, the probability of having seen an element of Y after

n steps, we define *the convergence number* of $(\mathcal{X}_i)_{i\in\mathbb{N}}$ with respect to Y by

$$N_{conv} = \min\left\{ n \,\middle|\, P\left(\{\mathcal{X}_j \,|j \leq n\} \cap Y \neq \emptyset\right) \geq \frac{1}{2} \right\}$$

In other words, it is the minimum number of steps such that the probability of having seen an element of Y is $\geq \frac{1}{2}$.

Remark 3.3.4. This captures the intuition of what it means for a sequence, and hence a transition procedure to be "better" than another: If $(\mathcal{X}_i)_{i\in\mathbb{N}}$ has a smaller convergence number than $(\mathcal{X}'_i)_{i\in\mathbb{N}}$, then it is "better."

The actual performance of an optimization method is given by its convergence number and by the average time it takes to perform one transition from $\mathcal{X}_{i-1}$ to $\mathcal{X}_i$. In this way, things factor out nicely between the "semantic aspect" (convergence number), and the "implementation aspect" (average speed).

It is also possible to imagine many other schemes which do not fall, strictly speaking in the framework of a single sequence of random variables, let alone a Markov chain. Such schemes are often called "Heuristics for Optimization." We mention the principal ones for the reader's interest in Appendix A, but they will not be used in this thesis. Given access to a super-computer, such schemes can be combined with those we have used to put many loosely connected bundles of processors to work together in non-trivial ways.

3.4 Generalization of Metropolis-Hastings

We now give the methods we shall use in Chapters 5 and 6 to implement fast algorithms to study Hadamard matrices. First, we define the general procedure for arbitrary selection and generation procedure (algorithm M), then we discuss two special choices for the selection procedure which can accommodate generation procedures producing an arbitrary number of candidates. In Section 5.1, we shall give a few generation procedures to find Hadamard matrices. However, the selection procedures we develop in Sections 3.4 and 3.5 are generally applicable.

Algorithm M (*Generalized Multi-Metro*[7]). Let X be a set, $f : X \to \mathbb{R}$ a function, $\mathcal{G} : X \to \cup_{n\in\mathbb{N}\setminus\{0\}} \left(X^{\Omega}\right)^n$ a generation procedure, and $\mathcal{S} : \cup_{n\in\mathbb{N}\setminus\{0\}} (\mathbb{R})^n \to (\mathbb{N}\setminus\{0\})^{\Omega}$ a selection procedure. Given x_0 and n, the algorithm visits the first n points in the induced Markov chain starting at x_0, for the transition procedure with energy f, generation $\mathcal{G}$, and selection $\mathcal{S}$. Furthermore, it produces x_{min}, γ_{min} according to the exhaustive search applied to the first $\mathcal{X}_i$, $i \in \{0, ..., n\}$ in the induced Markov chain and produces $x = \mathcal{X}_n, \gamma = f(\mathcal{X}_n)$. We would invoke it as $x_{min}, \gamma_{min}, x, \gamma \leftarrow \text{MultiMetro}(\mathcal{G}, \mathcal{S}, f, x_0, n)$.

M1. [Initialize.] Set $i \leftarrow 1, x \leftarrow x_0$, $\gamma \leftarrow f(x)$, $x_{min} \leftarrow x$, and $\gamma_{min} \leftarrow \gamma$.

M2. [Go through n steps.] Set $i \leftarrow i + 1$. If $i > n$, terminate.

M3. [Get the candidates.] Set $C \leftarrow (\!|\mathcal{G}(x)|\!)$.

M4. [Compute their energies.] $[\![\forall_i]\!]$ Set $\epsilon_i \leftarrow f(C_i)$. (In parallel.)

M5. [Select accordingly] Set $i_{selected} \leftarrow (\!|\mathcal{S}(\epsilon)|\!)$.

M6. [Fetch selected.] Set $x \leftarrow C_{i_{selected}}$ and $\gamma \leftarrow f(x)$.

M7. [New min?] If $\gamma_{min} > \gamma$, set $\gamma_{min} \leftarrow \gamma$, and $x_{min} \leftarrow x$.

M8. [Loop.] Go back to M2. ∎

Definition 3.4.1. Given $T > 0$, we define two selections.

1. the Satisficing *selection*

$$\mathcal{S}_{sat,T} : \cup_{n\in\mathbb{N}\setminus\{0\}} (\mathbb{R})^n \to (\mathbb{N}\setminus\{0\})^{\Omega}$$

such that if $(\epsilon_i)_{i=1}^n$ is a list of n elements of $\mathbb{R}$, then $\mathcal{S}_{sat,T}((\epsilon_i)_{i=1}^n)$ is a random variable distributed according to $\sum_{j\in \text{Sat}} \frac{1}{|\text{Sat}|}\delta_j$, where

$$\text{Sat} = \left\{ i \in \{2, .., n\} \,\middle|\, e^{-\epsilon_i/T} \geq R \right\}$$

[7] *This is not quite standard terminology. However, there are known special cases of this which are suitable for sampling from a given distribution, and those are called in the literature "Multiple-try Metropolis." For instance, see [31].*

and R is a random variable distributed uniformly in the interval $\left[0, e^{-\epsilon_1/T}\right)$. In other words, we set a threshold somewhere below $e^{-\epsilon_1/T}$, and select uniformly among the candidates j for which $e^{-\epsilon_j/T}$ is above or at the threshold.

2. the *Fair selection* $\mathcal{S}_{fair,T} : \cup_{n\in\mathbb{N}\setminus\{0\}} (\mathbb{R})^n \to (\mathbb{N}\setminus\{0\})^\Omega$ such that if $(\epsilon_i)_{i=1}^n$ is a list of n elements of $\mathbb{R}$, then $\mathcal{S}_{fair,T}\left((\epsilon_i)_{i=1}^n\right)$ is a random variable distributed according to $\sum_{i=1}^n p_i\delta_i$ with

$$p_i = \frac{e^{-\epsilon_i/T}}{\sum_j e^{-\epsilon_j/T}}$$

for every i. In other words, we pick $\mathcal{S}_{fair,T}\left((\epsilon_i)_{i=1}^n\right) = \kappa$ with odds $e^{-\epsilon_\kappa/T}$.

Remark 3.4.2. In the case of Satisficing selection, we see that ϵ_1 is treated differently than the other indices. If we ensure that the corresponding generation procedure $\mathcal{G}$ projected onto the first component is the identity, (namely that, for every x, $\mathcal{G}(x) = (x, \mathcal{G}'(x))$ for some other generation procedure $\mathcal{G}'$), then we see that Algorithm M will call $\mathcal{S}_{sat,T}$ with $\epsilon_1 = f(x)$, the current energy, and so the threshold will be set according to the current "Boltzmann factor" $e^{-f(x)/T}$.

Remark 3.4.3. Despite slight differences between these two selections, we have found during numerical experimentation, that, once T was well-chosen, they behaved very similarly. In order to simplify the following chapters, we shall use $\mathcal{S}_{fair,T}$ whenever both could be used more or less interchangeably.

However, conceptually, they are very different. For instance, the Fair selection discriminates between all of its candidates according to their Boltzmann factors, whereas the Satisficing selection can only ever discriminate between candidates whose energy is higher than the current energy (ϵ_1). This means that if the Fair selection ever sees a global minimum in its set of candidates, it will most likely select it, whereas the Satisficing selection would not be, at any single step, more likely to select a global minimum than any other candidate with an energy lower than the current one. This is where the name "Satisficing" came from [7]; as long as the threshold is met, there is absolutely no preference between a "low" energy and a "very low" energy.

For this reason, Fair selection is closer to Hill Climbing (Exhaustive Search for

the optimum among the candidates) than is Satisficing selection. However, when $T \to \infty$, $\mathcal{S}_{fair,T}$ become akin to Random Search, as the Boltzmann factors $\frac{e^{-\epsilon_i/T}}{\sum_{j=1}^{n} e^{-\epsilon_j/T}}$ all converge to the same value $(\frac{1}{n})$. Similarly for the Satisficing selection, every candidate becomes equiprobable when $T \to \infty$, because they are all at the threshold (which goes to 0).

Definition 3.4.4. If a transition procedure τ has energy E, generation procedure $\mathcal{G}$, and selection procedure $\mathcal{S}$ such that $\mathcal{S} = \mathcal{S}_{fair,T}$ is the Fair selection, we write $\tau = \text{Fair}_{T,\mathcal{G},E}$ and say that is satisfies the *Fair Multi-Metropolis semantics.*

Now that we have a precise target behavior (semantics) for the selection, we will show how to express it as a *reduction operation*, which is parallelizable.

We will denote $\mathbb{R}_+ = \{x \in \mathbb{R} \,|\, x \geq 0\}$ in what follows.

Algorithm SS (*Compute* $\mathcal{S}_{sat,T}$). Let SampleDiscreteOdds be such that if $V \in (X \times \mathbb{R}_+)^n$ with $V_i = (x_i, o_i)$ and $\sum_i o_i \neq 0$, then SampleDiscreteOdds (V) is a random variable distributed according to $\sum_{i=1}^{n} p_i \delta_{x_i}$ with $p_i = \frac{o_i}{\sum_j o_j}$. Given $\epsilon \in \mathbb{R}^n, T > 0$, we produce $i = \mathcal{S}_{sat,T}(\epsilon)$, unless none of the candidates pass the threshold, in which case the behavior depends on SampleDiscreteOdds. We would invoke it as $i \leftarrow \text{SelectSat}(\epsilon, T)$.

SS1. [Compute Threshold.] Set $u \leftarrow (\!|\text{Unif}|\!)$, and $R \leftarrow ue^{-\epsilon_1/T}$. (After, R is distributed uniformly in $\left[0, e^{-\epsilon_1/T}\right)$, where ϵ_1 has been arranged to be the current value of f.)

SS2. [Compute Odds.] $[\![\forall_j]\!]$, set $\mathsf{Pairs}_j \leftarrow \left(j, \begin{cases} 1 & e^{-\frac{\epsilon_j}{T}} \geq R \\ 0 & \text{Otherwise} \end{cases}\right)$. (After, Pairs represents the distribution $\sum_{j=1}^{N} \frac{\left[e^{-\epsilon_j/T} \geq R\right]}{\sum_{k=1}^{N}\left[e^{-\epsilon_k/T} \geq R\right]} \delta_j$, with $[\cdot]$ being 1 if $\cdot$ is true and 0 otherwise.)

SS3. [Sample.] Call $i \leftarrow (\!|\text{SampleDiscreteOdds}(\mathsf{Pairs})|\!)$. ∎

Similarly, we have:

Algorithm SF (*Compute* $\mathcal{S}_{fair,T}$). Let SampleDiscreteOdds be as above. Given $\epsilon \in \mathbb{R}^n, T > 0$, we produce $i = \mathcal{S}_{fair,T}(\epsilon)$. We would invoke it as $i \leftarrow \text{SelectFair}(\epsilon, T)$.

SF1. [Compute Odds.] $[\![\forall_j]\!]$, set $\mathsf{Pairs}_j \leftarrow \left(j, e^{-\epsilon_j/T}\right)$.

SF2. [Sample.] Call $i \leftarrow (\!|\text{SampleDiscreteOdds}\,(\mathsf{Pairs})|\!)$. ∎

Now we discuss SampleDiscreteOdds.

3.5 Finitely Supported Distribution Sampling as a Parallel Reduction

Definition 3.5.1. Let X be a set and let $Z = M(X) \times \mathbb{R}_+$ where $M(X)$ is the set of measures over X. We define a binary operation

$$\begin{aligned} \oplus : Z \times Z &\rightarrow Z \\ (\mu_1, o_1) \oplus (\mu_2, o_2) &= \begin{cases} \left(\frac{o_1}{o_1+o_2}\mu_1 + \frac{o_2}{o_1+o_2}\mu_2, o_1 + o_2\right) & o_1 + o_2 \neq 0 \\ \left(\frac{1}{2}\mu_1 + \frac{1}{2}\mu_2, 0\right) & o_1 + o_2 = 0 \end{cases} \end{aligned}$$

where we take the convention that we call $\oplus$ the *odds combination* operation. We can also define a binary operation on $Z' = X \times \mathbb{R}_+$ by identifying elements of X with Dirac measures:

$$\begin{aligned} \oplus : Z' \times Z' &\rightarrow Z \\ (x_1, o_1) \oplus (x_2, o_2) &= (\delta_{x_1}, o_1) \oplus (\delta_{x_2}, o_2) \end{aligned}$$

Proposition 3.5.2. *The odds combination operation is associative and commutative.*

Proof. Commutativity is obvious by the symmetry of the definition. Using the convention that, if $\forall_j o_j = 0$, then $\frac{o_i}{\sum_{j=1}^n o_j} = \frac{1}{n}$, we show associativity.

Let $(\mu_1, o_1), (\mu_2, o_2), (\mu_3, o_3) \in Z$. Then, $((\mu_1, o_1) \oplus (\mu_2, o_2)) \oplus (\mu_3, o_3)$ gives

$$\left(\frac{o_1 + o_2}{o_1 + o_2 + o_3}\left(\frac{o_1}{o_1 + o_2}\mu_1 + \frac{o_2}{o_1 + o_2}\mu_2\right) + \frac{o_3}{o_1 + o_2 + o_3}\mu_3, o_1 + o_2 + o_3\right)$$

and then

$$\left(\sum_i \frac{o_i}{\sum_j o_j}\mu_i, \sum_j o_j\right)$$

The same steps show that $(\mu_1, o_1)\oplus((\mu_2, o_2) \oplus (\mu_3, o_3))$ gives the same answer. $\square$

Remark 3.5.3. Associativity and commutativity in turn imply n-fold associativity and commutativity, as usual. Therefore, given $(z_i)_{i=1}^n \in Z^n$, we can write $\oplus_{i=1}^n z_i = z_1 \oplus z_2 \oplus ... \oplus z_n$ with any valid[8] arrangement of parentheses without changing the final value.

In what follows, Π_i will denote the projection onto the i-th component of a tuple.

Theorem 3.5.4. *(Finite Support Sampling by Reduction[9]). Let $\mathcal{D} = \sum_{i=1}^n p_i\delta_{x_i}$ any probability measure over X with finite support. Then, we have $\mathcal{D} = \Pi_1 (\oplus_{i=1}^n (x_i, o_i))$ for any list of positive numbers $o \in \mathbb{R}_+^n$ such that there exists $\alpha \in \mathbb{R}_+$ with $o_i = \alpha p_i$ for every i.*

In other words, we can sample from any finitely supported distribution simply by knowing, up to a constant, the probabilities on that support, by a reduction.

Remark 3.5.5. Though this result is immediate from the definitions using induction, it has profound implications. Indeed, reductions can be performed in parallel. See for instance [22] for the practicalities of implementing reduction on a GPU (Graphical Processing Unit).

We give algorithms B and LB as references, but they are merely some of the well-known parallel reduction algorithms. Furthermore, their performance characteristics are easily seen and also well-known.

We start with the simplest.

[8] In the usual sense, namely that a valid parenthesized reduction is either $(\alpha \oplus \beta)$ with both α and β being smaller valid parenthesized reductions, or it is z_i with $i \in \{1, ..., n\}$.

[9] *Despite the simplicity of this result, the author hasn't been able to find mention of it in the literature. It seems hard to believe that such a simple extension of the concept of a parallel summation, which was known already in 2007, hasn't been discovered in all of that time. Therefore, we apologize if there indeed is such a result in published form or if this is known in practice, of course the credit goes to whomever found it first.*

Algorithm B (*Binary reduction*). Let Z a set, $\boxplus : Z \times Z \to Z^{\Omega}$ a (possibly random) binary operation, and $V \in Z^n$. Given $\boxplus, V, n$, we produce $z = \boxplus_{i=0}^{n-1} V_{[in]i}$, and destroy $V_{[in]}$. We would invoke it as $z \leftarrow \text{BinaryRedux}(\boxplus, V, n)$.

B1. [Power of two?] First set $m \leftarrow \lfloor \log_2(n) \rfloor$. If $n \neq 2^m$, $[\![\forall_{j \in \{2^m, 2^m+1, \ldots, n-1\}}]\!]$, set $V_{j-2^m} \leftarrow (\!| V_{j-2^m} \boxplus V_j |\!)$. (Here, $\lfloor x \rfloor$ is the biggest integer which is $\leq x$. After this step, $\boxplus_{i=0}^{2^m-1} V_i = \boxplus_{i=0}^{n-1} V_{[in]i}$.)

B2. [Reduce by two.] If $m = 0$, set $z \leftarrow V_0$ and terminate.

B3. [Half of remaining operations.] $[\![\forall_{j \in \{2^{m-1}, 2^{m-1}+1, \ldots, 2^m-1\}}]\!]$, set $V_{j-2^{m-1}} \leftarrow (\!| V_{j-2^{m-1}} \boxplus V_j |\!)$. (After this step, $\boxplus_{i=0}^{2^{m-1}-1} V_i = \boxplus_{i=0}^{n-1} V_{[in]i}$.)

B4. [Loop.] Set $m \leftarrow m - 1$, and go back to B2. ∎

Lemma 3.5.6. *If we count one application of $\oplus$ as one operation, then there exists a parallel algorithm which takes as inputs a list $\mathsf{Pairs}_{[in]} \in (X \times \mathbb{R}_+)^n$, and outputs $\Pi_1 \left(\oplus_{i=1}^{n} \mathsf{Pairs}_{[in],i} \right)$ with a work $\Upsilon \in \Theta(n)$, a span $\Upsilon_\infty \in \Theta(\log_2(n))$, and a parallelism $S_\infty \in \Theta\left(\frac{n}{\log_2(n)} \right)$. (See algorithm B.)*

Furthermore there exists an algorithm with all above characteristics such that the parallelism is linearly accessible. (See algorithm LB below.)

Algorithm LB (*Linear and Binary reduction*). Let $\boxplus : Z \times Z \to Z^{\Omega}$ and $V \in Z^n$ as in algorithm B, and let p be the currently available number of processors. Given $\boxplus, V, n, p$, we produce $z = \boxplus_{i=0}^{n-1} V_{[in]i}$, and destroy $V_{[in]}$. We would invoke it as $z \leftarrow \text{LinBinRedux}(\boxplus, V, n, p)$.

LB1. [Processors $<$ objects?] If $p \geq n$, go to LB6.

LB2. [Remainder?] First set $m \leftarrow \left\lfloor \frac{n}{p} \right\rfloor$, and $r \leftarrow n - pm$. If $r \neq 0$, $[\![\forall_{j \in \{pm, \ldots, n-1\}}]\!]$, set $V_{j-p} \leftarrow (\!| V_{j-p} \boxplus V_j |\!)$. (After this step, $\boxplus_{i=0}^{pm-1} V_i = \boxplus_{i=0}^{n-1} V_{[in]i}$.)

LB3. [To binary.] If $m = 1$, go to LB6.

LB4. [Do p operations.] $[\![\forall_{j \in \{p(m-1), \ldots, pm-1\}}]\!]$, set $V_{j-p} \leftarrow (\!| V_{j-p} \boxplus V_j |\!)$. (After this step, $\boxplus_{i=0}^{p(m-1)-1} V_i = \boxplus_{i=0}^{n-1} V_{[in]i}$.)

LB5. [Loop.] Set $m \leftarrow m - 1$, and go back to LB3.

LB6. [Binary reduction.] Call $z \leftarrow \text{BinaryRedux}(\boxplus, V, \min(n, p))$. ∎

With this, we can come back to the selection methods, and discuss their performance.

Proposition 3.5.7. *Algorithms SS and SF (for computing the Satisficing and Fair selections), when implemented with algorithm LB for the finite support sampling, both have work* $\Theta(n)$*, span* $\Theta(\log_2(n))$*, parallelism* $\Theta\left(\frac{n}{\log_2(n)}\right)$*. Furthermore, this parallelism, in both cases, is linearly accessible.*

Proof. For algorithm SS, we compute the work, span, and linear accessibility for each step:

- For SS1, the work is $\Theta(1)$, span $\Theta(1)$, and the parallelism is linearly accessible.
- For SS2, the work is $\Theta(n)$, span $\Theta(1)$, linearly accessible.
- For SS3, all follows algorithm LB, so work $\Theta(n)$, span $\Theta(\log_2(n))$, linearly accessible.

Combining according to Lemma 2.5.4, we get the result.

Similarly for algorithm SF. □

Remark 3.5.8. It turns out there are some (asymptotically) efficient methods for sampling serially from a finitely-supported distribution. However, such methods still require a linear number of steps to "set-up," and then, if we generate many random samples from the same distribution, then we can do so in constant time, see [30]. This is quite interesting, but for us, we want sublinear span and we want no "setting-up" since we only use a given distribution once. Furthermore, we don't want the work to be greater than it was originally. Finally, we need this to have good performance even for small values of m. Therefore, parallel reduction must beat any single-sample serial techniques, since such a technique has to at least look at all the odds, and so the work must be the same as for our reduction.

Notice however, that the tables are turned if we must sample many times from a fixed distribution. Indeed, in such a case, it would be better to take the "set-up" cost, and then simply sample according to the alias method, or some other method, for every available processor. Then, instead of producing 1 sample in span $\Theta(\log_2(n))$, we would produce n samples in span $\Theta(1)$ (ignoring the set-up costs).

3.6 Gradient Methods

Now that we have seen methods of optimization which make almost no assumptions about the underlying space, we briefly discuss some that *do* make assumptions. For instance, that the function to optimize be differentiable, or that the gradient be computable using a closed formula. Notice however, that the underlying space X now is allowed to be isomorphic to $\mathbb{R}^N$ for some N. In particular it is uncountable, with many degrees of freedom. It isn't surprising that we will assume differentiability, or at least continuity in such a case.

But in the trade between finiteness and differentiability, we end up with a quite different problem.

We have not made use of these so-called "gradient" methods in our thesis (having mainly focused on real Hadamard matrices, contained in the finite set $\mathcal{M}_m(\pm)$), but we have devised fast algorithms to exploit such methods in the search for complex Hadamard matrices for two reasons.

1. Developing our algorithms in a generic context revealed and clarified the mathematical structure of concepts which seemed at first to only be implementation details.

2. Some research focuses on the field [46] of complex Hadamard matrices, and many questions are still open in single and double-digits dimensions. So the improvements of the speed of our algorithms, which are currently not enough to surmount the cases of $m = 668$ and above, could well reap much success in the complex case, for dimensions such as 11 or 16. This is one of the main avenues for future work.

We only cover the simplest gradient method (Gradient Descent), as the algorithmic progress we have made is in the computation of the gradient itself, mostly, as well as in the computation of partial stationary points. Thus, our gradient computations could be adapted straightforwardly to serve more sophisticated gradient methods such as Conjugate Gradient Descent (see [34, pp. 99-105]).

Remark 3.6.1. Note that $\mathcal{M}_m(\mathbb{R}) \cong \mathbb{R}^{m^2}$ by the transformation of indices from (i,j) to $(i-1)m+j$. More precisely, a function $f : \{1,...,m^2\} \to \mathbb{R}$, would be sent to a matrix $M \in \mathcal{M}_m(\mathbb{R})$ given by $[M]_{ij} = f((i-1)m+j)$. In what follows, we recall concepts of multi-variable calculus, which are typically presented for $\mathbb{R}^n$, but emphasize that because of the isomorphism, these concepts translate directly to $\mathcal{M}_m(\mathbb{R})$, which is how we shall make use of them in Section 5.1.

Definition 3.6.2. Letting $f : \mathbb{R}^n \to \mathbb{R}$, $x \in \mathbb{R}^n$, for every $i \in \{1,...,n\}$, we define the *partial function* $\text{part}_{f,x,i} : \mathbb{R} \to \mathbb{R}$ of f at x in the direction i by the formula

$$\text{part}_{f,x,i}(\theta) = f(x + \theta e_i)$$

where, $[e_i]_k = \begin{cases} 1 & k = i \\ 0 & k \neq i \end{cases}$, is the i-th vector in the canonical basis of $\mathbb{R}^n$.

Furthermore, if the partial functions are differentiable in a neighborhood of x, then the *gradient* $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ for a given $x \in \mathbb{R}^n$, will satisfy

$$[(\nabla f)(x)]_i = \frac{d}{d\theta}\text{part}_{f,x,i}|_{\theta=0}$$

for $1 \leq i \leq n$.

Remark 3.6.3. The gradient is used in many optimization algorithms for differentiable functions over $\mathbb{R}^n$, since it points in the direction for which the change in the function is most steep.

Definition 3.6.4. The set of *stationary points* for a differentiable function $f : \mathbb{R} \to \mathbb{R}$ is

$$\text{stat}_f = \left\{ x \in \mathbb{R} \,\middle|\, \frac{d}{d\theta} f|_{\theta=x} = 0 \right\}$$

Proposition 3.6.5. *A bounded, differentiable, and periodic function $f : \mathbb{R} \to \mathbb{R}$ has a global minimum, whose preimage is contained within its stationary points.*

Remark 3.6.6. The proof is elementary. Notice that differentiability is not necessary (can be replaced by continuity) to show that f has a global minimum. Furthermore,

the periodicity and the boundedness are not necessary to show that all local minima are stationary points. But since stationary points shall only be considered for bounded differentiable periodic functions in Section 5.1, these truths have been merged for convenience.

Proposition 3.6.5 justifies why we might care about the partial stationary points (defined below). Indeed, the global minima of partial functions will occur within the set of stationary points of those partial functions.

Definition 3.6.7. Let $f : \mathbb{R}^n \to \mathbb{R}$, $x \in \mathbb{R}^n$, i such that $\text{part}_{f,x,i}$ is differentiable, we define the partial stationary points of f at x in the direction i as the set

$$\text{Stat}_i\left(f, x\right) = \left\{x + \theta e_i | \theta \in \text{stat}_{\text{part}_{f,x,i}}\right\}$$

(If θ is a stationary point of $\text{part}_{f,x,i}$, then $x + \theta e_i$ is a partial stationary point of f.)

We define the gradient descent as a transition procedure

Definition 3.6.8. Let $X = \mathbb{R}^n$, $f \in \mathbb{R}^X$ a differentiable function, and $\delta > 0$ a "step size". We define the *gradient descent transition procedure*

$$\begin{aligned} \tau_{Descent,f,\delta} : X &\to X \\ v &\mapsto v - \delta\left(\nabla f\right)\left(v\right) \end{aligned}$$

For $0 \leq \epsilon \leq 1$, we say that τ is an ϵ-sparsified gradient descent transition procedure if there exists $G : X \to X$ such that, for all $v \in X$, the following hold

$$\begin{aligned} \tau\left(v\right) &= v - \delta G\left(v\right) \\ \forall_i . \left[G\left(v\right)\right]_i &\in \left\{0, \left[\left(\nabla f\right)\left(v\right)\right]_i\right\} \\ \left\|G\left(v\right)\right\|^2 &\geq \epsilon \left\|\left(\nabla f\right)\left(v\right)\right\|^2 \end{aligned}$$

where $\|v\|$ denotes the euclidean norm $\sqrt{\sum_i |v_i|^2}$.

In other words, G is a version of ∇f with some components replaced by 0, such

that the euclidean norm is not too diminished (the information about ∇f isn't all lost).

If τ is an ϵ-sparsified gradient descent transition procedure, we can say $\tau \in \tau_{Descent,f,\delta,\epsilon}$, where $\tau_{Descent,f,\delta,\epsilon}$ is simply the set of all such functions.

Proposition 3.6.9. *For f, δ as above, and $\epsilon_1 \leq \epsilon_2$, we have*

1. $\tau_{Descent,f,\delta,\epsilon_2} \subseteq \tau_{Descent,f,\delta,\epsilon_1}$
2. $\tau_{Descent,f,\delta,1} = \{\tau_{Descent,f,\delta}\}$
3. $\tau_{Descent,0,\delta} \in \tau_{Descent,f,\delta,0}$

Proof. This follows directly from the definition. With all the notation as above, for $\tau \in \tau_{Descent,f,\delta,\epsilon_2}$, there must exist G satisfying, for all v,

$$\begin{aligned} \tau(v) &= v - \delta G(v) \\ \forall_i. [G(v)]_i &\in \{0, [(\nabla f)(v)]_i\} \\ \|G(v)\|^2 &\geq \epsilon_2 \|(\nabla f)(v)\|^2 \end{aligned}$$

Since $\|G(v)\|^2 \geq \epsilon_2 \|(\nabla f)(v)\|^2$ and $\epsilon_2 \geq \epsilon_1$, we have $\|G(v)\|^2 \geq \epsilon_1 \|(\nabla f)(v)\|^2$, and so $\tau \in \tau_{Descent,f,\delta,\epsilon_1}$.

Similarly, if $\epsilon = 1$, we get

$$\begin{aligned} \tau(v) &= v - \delta G(v) \\ \forall_i. [G(v)]_i &\in \{0, [(\nabla f)(v)]_i\} \\ \|G(v)\|^2 &\geq \|(\nabla f)(v)\|^2 \end{aligned}$$

which implies $\forall_i. [G(v)]_i = [(\nabla f)(v)]_i$. The converse is obvious.

Finally, taking $[G(v)]_i = 0, \forall_i$, corresponding to $G = \nabla 0$ (for instance), will satisfy both $\|G(v)\|^2 \geq 0$ (trivially) and $\forall_i. [G(v)]_i \in \{0, [(\nabla f)(v)]_i\}$. □

Remark 3.6.10. Note that the G in the above definition, which we informally think of as a "sparsified" version of the gradient, is a special case of a so-called *subgradient*. For more on subgradients and the related methods of "convex optimization," see [6].

Chapter 4

Hadamard matrices

4.1 Basic Properties of Hadamard matrices

In the quest to find Hadamard matrices, one sees that simply-described constraints can be very difficult to satisfy, and hold much mystery. In this section, we recall the definition of Hadamard matrices and review some of their properties. (See for instance [49].)

For $m \in \mathbb{N}\backslash\{0\}$, we will denote the set of "plus and minus one matrices" by

$$\mathcal{M}_m(\pm) = \left\{ M \in \mathcal{M}_m(\mathbb{R}) \,\middle|\, \forall_{i,j}.\, [M]_{ij} \in \{-1, +1\} \right\}$$

Definition 4.1.1. A matrix $H \in \mathcal{M}_m(\pm)$ is a *(real) Hadamard matrix* if $HH^T = m\mathbb{I}$, where H^T denotes the transpose of H. We also write $\mathcal{H}_m$ for the set of all real Hadamard matrices in $\mathcal{M}_m(\pm)$.

Proposition 4.1.2. *For $M \in \mathcal{M}_m(\pm)$, the following are equivalent:*

1. $M \in \mathcal{H}_m$
2. $M^T \in \mathcal{H}_m$
3. *The rows of M form an orthogonal set.*
4. *The columns of M form an orthogonal set.*

Proof. We see that 1. and 3. are respectively the matrix form or the component-by-component form of $MM^T = m\mathbb{I}$. Similarly, 2. and 4. are both equivalent to $M^T M = m\mathbb{I}$.

To see that 1. and 2. are equivalent, and thus connect all four propositions, we can see that $0 \neq m^m = \det(m\mathbb{I}) = \det\left(MM^T\right) = (\det(M))^2$ and thus M is invertible, so $\left(\frac{1}{m}\right) MM^T = \mathbb{I}$ implies $M^T = \left(\frac{1}{m}M\right)^{-1}$, and therefore $M^T\left(\frac{1}{m}M\right) = \mathbb{I}$, which gives $M^T\left(M^T\right)^T = m\mathbb{I}$. □

Proposition 4.1.3. *If $\mathcal{H}_m \neq \emptyset$, and $m > 2$, then 4 divides m.*

Proof. This is a standard result [49], but we sketch the proof. Let $m > 2$ such that $\mathcal{H}_m \neq \emptyset$. Choose $H \in \mathcal{H}_m$. Without loss of generality[1], we can choose H such that the first row is identically 1.

Call the first three rows u, v, w. The orthogonality constraints between u and w and between v and w give us that

$$\begin{aligned} \sum_i w_i &= 0 \\ \sum_{v_i=1} w_i - \sum_{v_i=-1} w_i &= 0 \end{aligned}$$

By adding these, we get

$$\begin{aligned} 2\sum_{v_i=1} w_i &= 0 \\ \sum_{\substack{v_i=1 \\ w_i=1}} 1 - \sum_{\substack{v_i=1 \\ w_i=-1}} 1 &= 0 \end{aligned}$$

Hence, the size of $\{i \,|v_i = 1 \text{ and } w_i = 1\}$ is exactly half of that of $\{i \,|v_i = 1\}$.

[1] If the initial choice isn't thus, simply use proposition 4.1.5 to multiply the relevant columns by -1 until all begin with $+1$, at which point the first row will be as before. Also note that a similar proof can be written without using multiplication by -1, except that instead of comparing elements to ± 1, we compare them to u_i.

Finally, the orthogonality constraint between u and v gives

$$\sum_{v_i=1} 1 - \sum_{v_i=-1} 1 = 0$$

So the size of $\{i \,|v_i = 1\}$ is exactly half of m. Hence $|\{i \,|v_i = 1 \text{ and } w_i = 1\}| = \frac{m}{4}$, and this must be an integer. □

The converse is still open and was conjectured by J. Hadamard in 1893.

Conjecture. *For any m, multiple of 4, $\mathcal{H}_m \neq \emptyset$.*

Let us introduce another characterization of Hadamard matrices. First, we recall the *Hadamard Inequality:* If $A = (a_1, a_2, ..., a_m) \in \mathcal{M}_m(\mathbb{R})$, then

$$|\det(A)| \leq \prod_{i=1}^{n} \|a_i\|$$

Moreover, we have equality if and only if the columns of A are orthogonal.

Proposition 4.1.4. *For $M \in \mathcal{M}_m(\pm)$, we have*

1. $|\det(M)| \leq m^{m/2}$
2. $M \in \mathcal{H}_m$ *if and only if* $|\det(M)| = m^{m/2}$

Proof. For the inequality, if $A = (a_1, a_2, ..., a_m) \in \mathcal{M}_m(\pm)$, then the result follows because $\|a_i\| = m^{\frac{1}{2}}$ for $1 \leq i \leq m$.

The second part follows similarly from the Hadamard Inequality. □

Recall that to any permutation $\sigma \in \mathcal{S}_m$ is associated a permutation matrix $P_\sigma \in \mathcal{M}_m(\mathbb{R})$ defined by

$$[P_\sigma]_{ij} = \begin{cases} 1 & j = \sigma(i) \\ 0 & j \neq \sigma(i) \end{cases}$$

We will denote by $\mathcal{P}_m$ the group of $(m \times m)$-permutation matrices and by $\mathcal{D}_m(\pm)$ the diagonal $(m \times m)$-matrices $\text{Diag}(d_1, d_2, ..., d_m)$ with $d_i \in \{-1, +1\}$. Note that for

$P \in \mathcal{P}_m$ and $D \in \mathcal{D}_m(\pm)$, $|\det(P)| = |\det(D)| = 1$. Then by Proposition 4.1.4, we have

Proposition 4.1.5. *With the above notations, $\mathcal{P}_m$ and $\mathcal{D}_m(\pm)$ are left and right multipliers of $\mathcal{H}_m$.*

(i.e. $\mathcal{P}_m\mathcal{H}_m$, $\mathcal{D}_m(\pm)\mathcal{H}_m$, $\mathcal{H}_m\mathcal{P}_m$, and $\mathcal{H}_m\mathcal{D}_m(\pm)$ are contained in $\mathcal{H}_m$.)

We denote by G_m the subgroup of $\mathrm{GL}(m, \mathbb{R})$ generated by $\mathcal{P}_m$ and $\mathcal{D}_m(\pm)$ and it is the semi-direct product of $\mathcal{D}_m(\pm)$ by $\mathcal{P}_m$. As a consequence of Proposition 4.1.5, we get an action of $G_m \times G_m$ on $\mathcal{H}_m$, given by

$$(g_1, g_2) M = g_1 M g_2^{-1}$$

This action induces an equivalence relation over $\mathcal{H}_m$ given by

$$H_1 \equiv H_2 \iff \exists_{(g_1,g_2)\in G_m\times G_m}.\, (g_1, g_2) H_1 = H_2$$

Then the cardinality of $\mathcal{H}_m/(G_m \times G_m)$ is the number of inequivalent Hadamard matrices in dimension m. Such numbers are known for $m \leq 32$ and unknown for $m > 32$ (when m is a multiple of 4), with the case $m = 32$ only recently proven [25] to be

$$|\mathcal{H}_m/(G_m \times G_m)| = 13\,710\,027$$

4.2 Constructions of Hadamard matrices

Even if the Hadamard Conjecture is still open, many different constructions of Hadamard matrices have been found. Indeed, the existence of Hadamard matrices for $m = 4n$ has been proven [21] for almost all dimensions below 1000. (There are four exceptions, the first being $m = 668$.) The publication [49] gives a nice survey of the history of the subject.

The first constructions were given by Sylvester, for instance

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \in \mathcal{H}_2 \tag{3}$$

Another example comes from the *Kronecker product* [40]. Recall that for two matrices $A \in \mathcal{M}_m(\mathbb{C})$ and $B \in \mathcal{M}_n(\mathbb{C})$, their Kronecker product $A \otimes B \in \mathcal{M}_{mn}(\mathbb{C}) \cong \mathcal{M}_m(\mathcal{M}_n(\mathbb{C}))$ is the matrix C defined by

$$[C]_{ij} = [A]_{ij} B$$

for all $1 \leq i, j \leq m$.

The following properties of the Kronecker product are well known. If $\alpha \in \mathbb{C}$, $A, A_1, A_2 \in \mathcal{M}_m(\mathbb{C})$ and $B, B_1, B_2 \in \mathcal{M}_n(\mathbb{C})$ then we have the following

$$(A \otimes B)^T = A^T \otimes B^T \tag{4}$$

$$(A_1 \otimes B_1)(A_2 \otimes B_2) = (A_1 A_2) \otimes (B_1 B_2) \tag{5}$$

$$(\alpha A) \otimes B = A \otimes (\alpha B) = \alpha (A \otimes B) \tag{6}$$

$$\mathbb{I}_m \otimes \mathbb{I}_n = \mathbb{I}_{mn} \tag{7}$$

We then have

Proposition 4.2.1. *The Kronecker product of two Hadamard matrices is itself a Hadamard matrix.*

By (3), we get

Corollary 4.2.2. *For all $n \geq 1$, $\mathcal{H}_{2^n} \neq \emptyset$.*

We now show a few examples[2] of known Hadamard matrices. To help visualization, we represent a matrix as a checker pattern. Every element of our matrix becomes a

[2]For the source of our examples, see the databases of Hadamard matrices at [21, 43]. Note, however, that the visualization pipeline that produced the figures in this thesis was programmed using the C programming language and the MetaPost system.

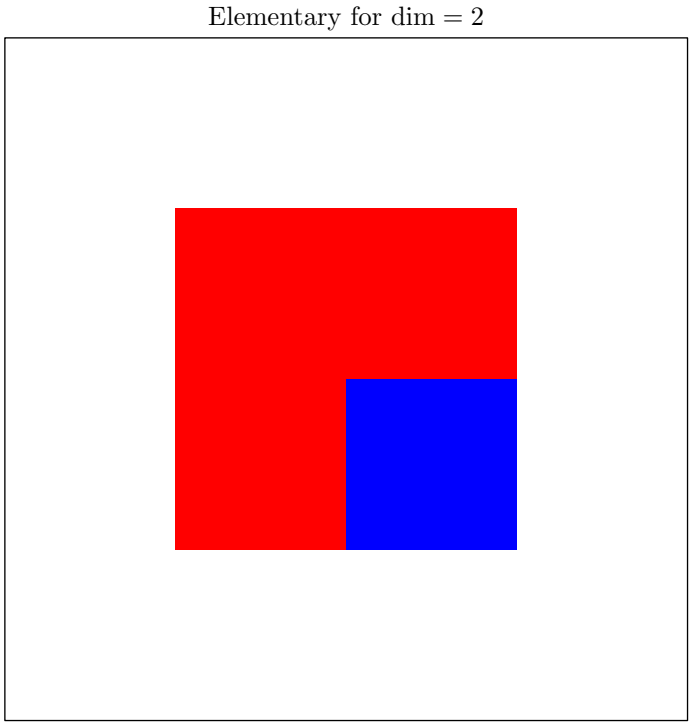


Figure 4.1: Visual representation of ± 1 matrices; H_2

colored square. By convention, "blue" is negative, and "red" is positive. For Hadamard matrices, blue is -1 and red is $+1$.

For instance, (3) is shown in figure 4.1, and other examples follow in figures 4.2 up to 4.6.

4.3 Complex Hadamard matrices

Looking at Definition 4.1.1, we can see a possible generalization, namely the complex Hadamard matrices. For an excellent presentation of complex Hadamard matrices, see [46].

In what follows, we will denote by $\mathbb{T}$ the unit circle in $\mathbb{C}$ and by C_q the q-th roots of unity.

Definition 4.3.1. A matrix $H \in \mathcal{M}_m(\mathbb{T})$ is a *complex Hadamard matrix* if $HH^\star = m\mathbb{I}$, where $H^\star$ denotes the adjoint of H. We also write $\mathcal{H}_m(\mathbb{T})$ for the set of all

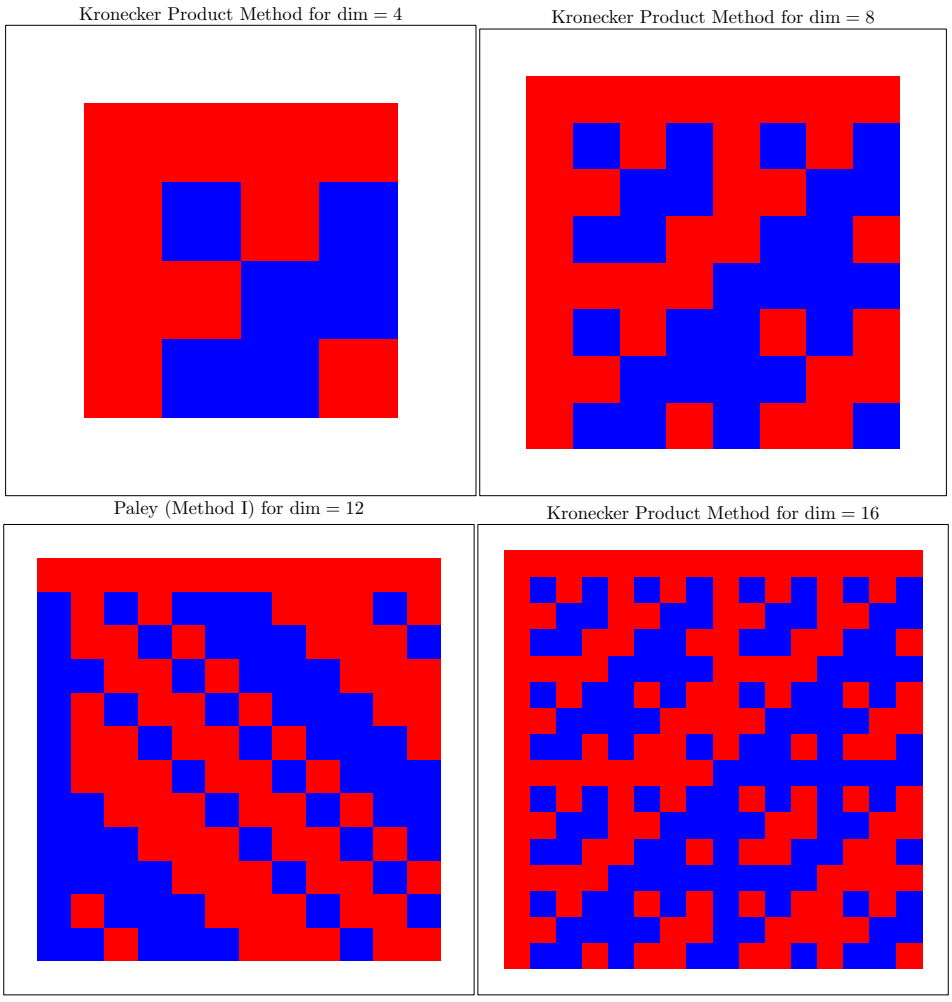


Figure 4.2: Examples of Hadamard matrices in dimensions 4, 8 (top), 12, and 16 (bottom).

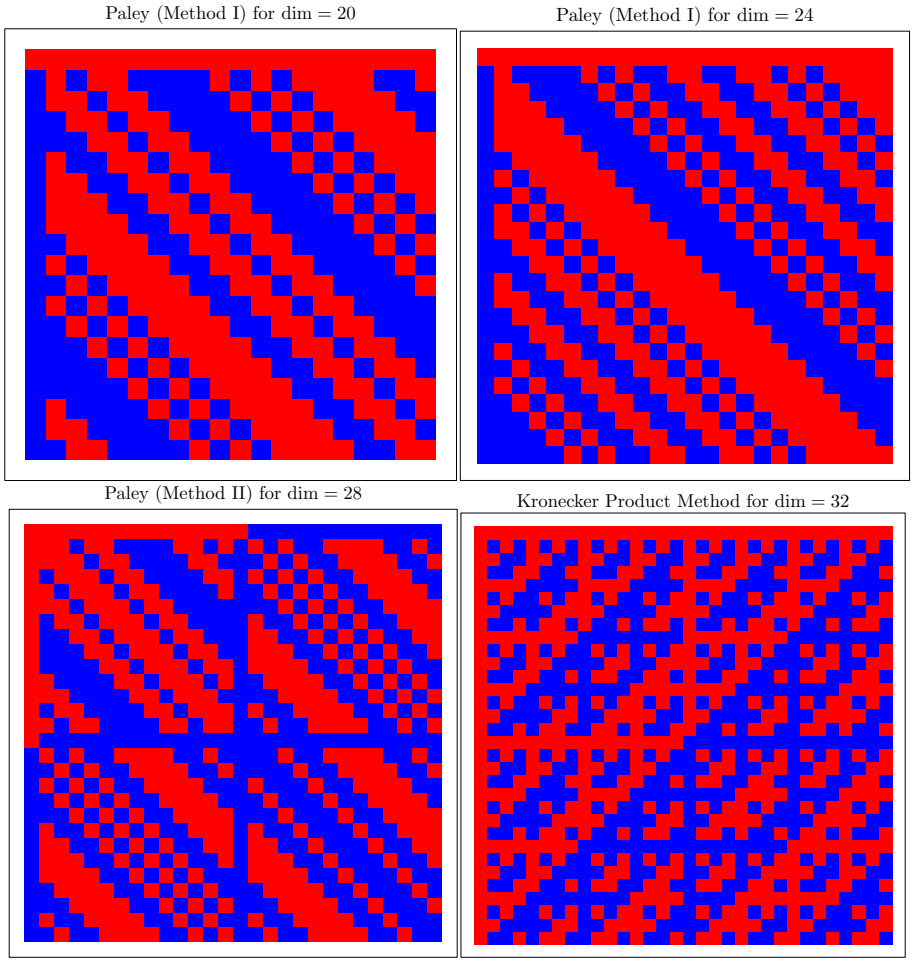
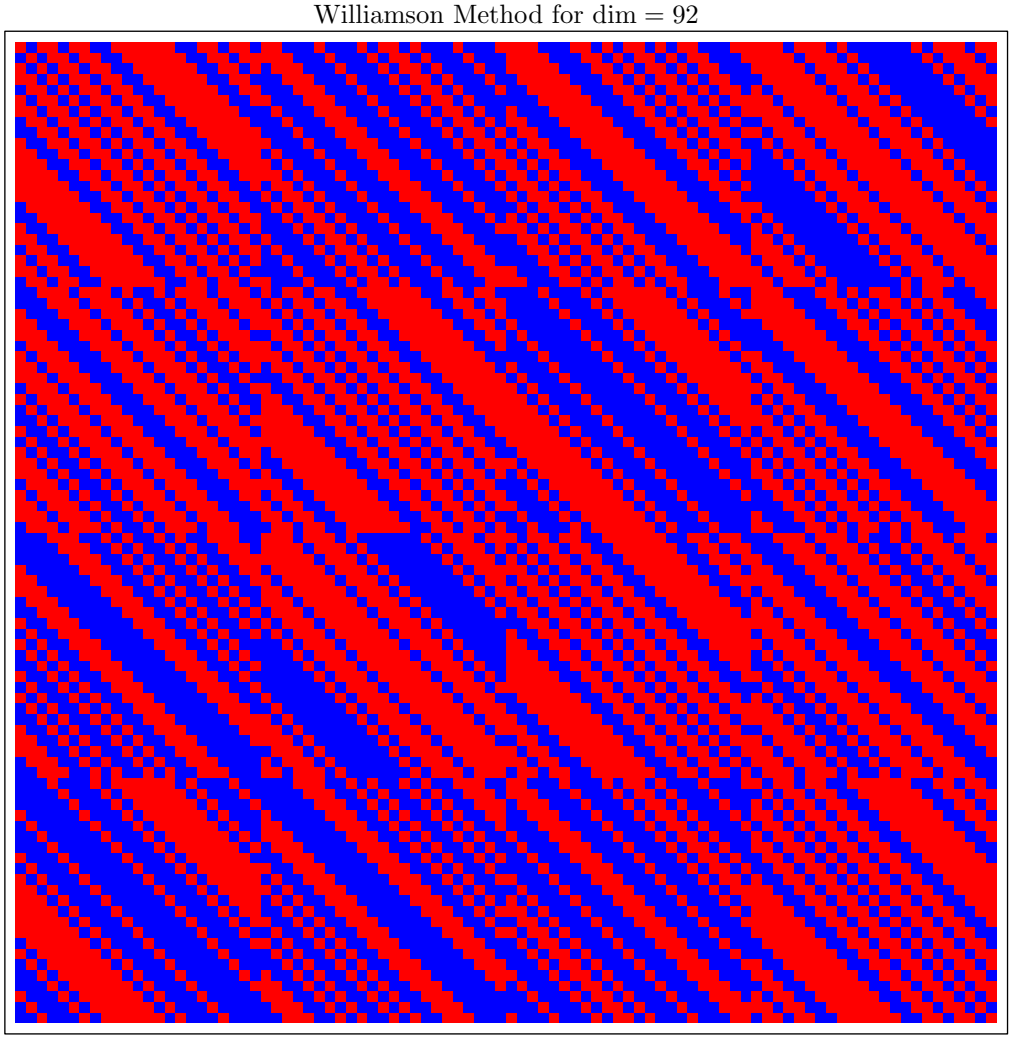


Figure 4.3: Hadamard examples: dim 20 to 32.



It took many years before humanity discovered a Hadamard matrix at dimension 92.

Figure 4.4: Hadamard Example dimension 92.

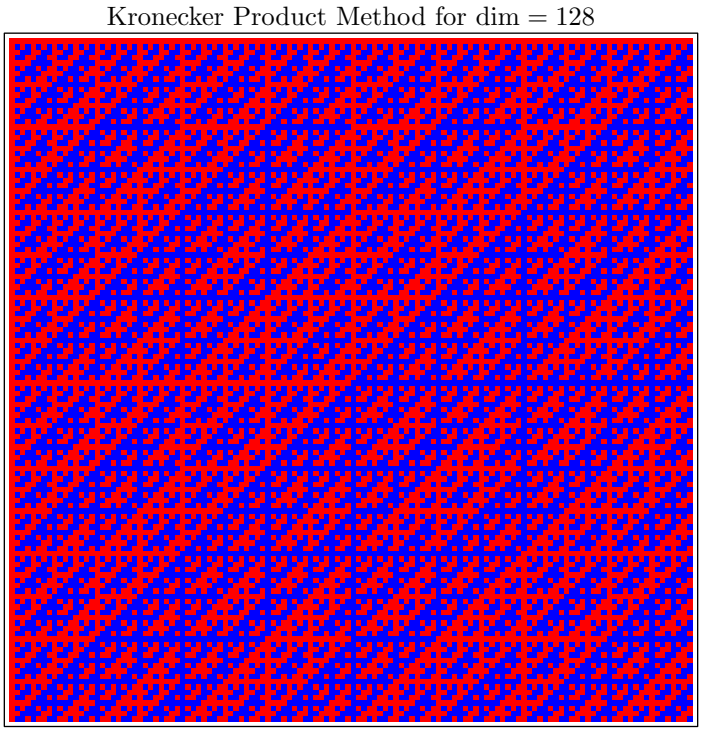


Figure 4.5: Example for dim = 128.

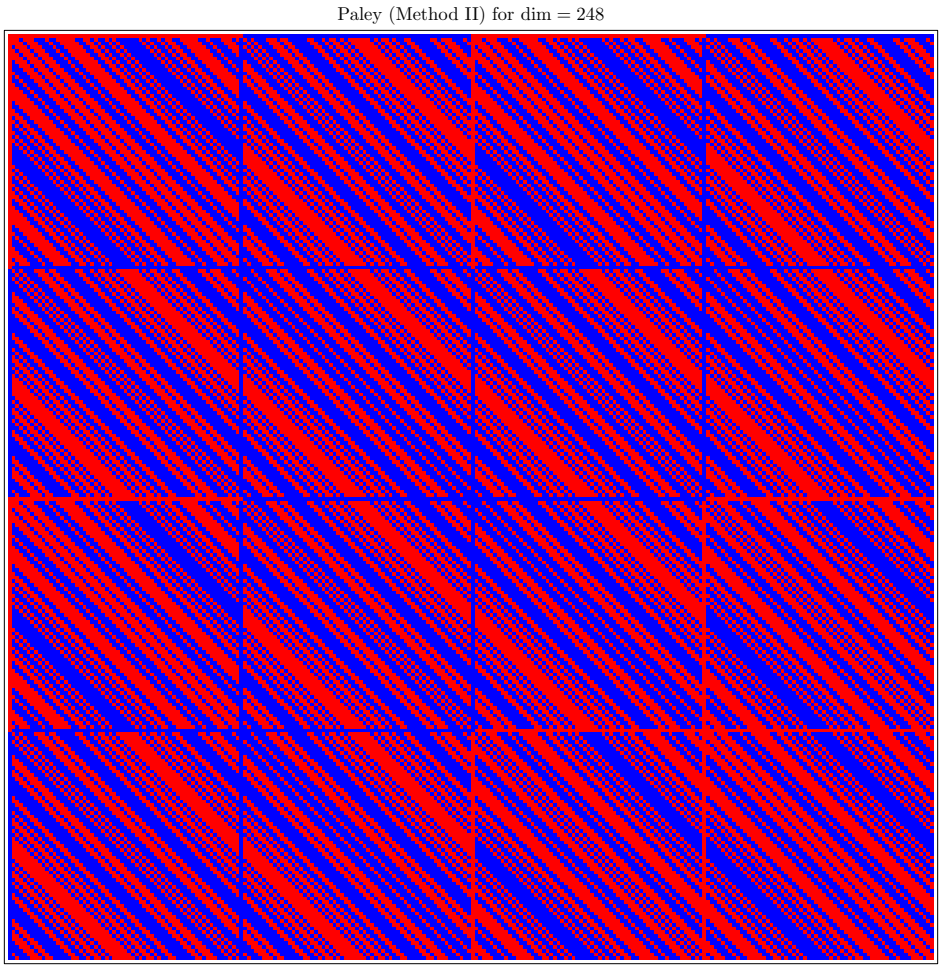


Figure 4.6: Example for dim = 248.

complex Hadamard matrices.

Letting $q \in \mathbb{N} \setminus \{0\}$, we also define the set of Hadamard matrices of Butson-type q as $\mathcal{H}_m(C_q) = \mathcal{H}_m(\mathbb{T}) \cap \mathcal{M}_m(C_q)$.

Note that $\mathcal{H}_m$ is just the special case $\mathcal{H}_m(C_q)$ with $q = 2$.

4.4 Finding Hadamard matrices as an Optimization Problem

We now combine Chapters 3 and 4. More precisely, we characterize $\mathcal{H}_m(\mathbb{T})$ with various optimization problems.

Definition 4.4.1. We define various functions from $\mathcal{M}_m(\mathbb{C})$ to $\mathbb{R}$. Let E_{dev}, E_{det}, E_{det^2}, E_{rows}, and E_{col} be such functions given by .

$$\begin{aligned}
E_{dev}(A) &= \sum_i \sum_j \left(\left| [A]_{ij} \right| - 1 \right)^2 \\
E_{det}(A) &= -\left| \det(A) \right| \\
E_{det^2}(A) &= -\left| \det(A) \right|^2 \\
E_{rows}(A) &= \sum_j \sum_{i<j} \left| [AA^\star]_{ij} \right|^2 \\
E_{col}(A) &= \sum_j \sum_{i<j} \left| [A^\star A]_{ij} \right|^2 \\
E_{Quasi}(A) &= \frac{1}{2(m-1)} \sum_j \sum_{i<j} \left| [AA^\star]_{ij} \right|
\end{aligned}$$

Also note that if $\|\cdot\|$ is a norm over $\mathbb{C}^{\frac{m(m-1)}{2}}$, we can define functions

$$E_{rows,\|\cdot\|}(A) = \left\| \left([AA^\star]_{ij} \right)_{i<j} \right\|$$

and

$$E_{cols,\|\cdot\|}(A) = \left\| \left([A^\star A]_{ij} \right)_{i<j} \right\|$$

where $\left([AA^\star]_{ij}\right)_{i<j}$ and $\left([A^\star A]_{ij}\right)_{i<j}$ are regarded as vectors in $\mathbb{C}^{\frac{m(m-1)}{2}}$.

This gives

$$\begin{aligned} E_{rows}(A) &= \left(E_{rows,\|\cdot\|_2}(A)\right)^2 \\ E_{cols}(A) &= E_{rows}(A^\star) \\ E_{Quasi}(A) &= \frac{1}{2(m-1)} E_{rows,\|\cdot\|_1}(A) \end{aligned}$$

Let us denote

$$O_m = \{M \in \mathcal{M}_m(\mathbb{C}) \,|\, MM^\star = m\mathbb{I}\}$$

With this we can characterize $\mathcal{H}_m$ in many ways.

Proposition 4.4.2. *For all m and all norms $\|\cdot\|$ over $\mathbb{C}^{\frac{m(m-1)}{2}}$ we have that*

$$\begin{gathered} (O_m, E_{dev}, 0) \\ \left(\mathcal{M}_m(\mathbb{T}), E_{det}, -m^{m/2}\right) \\ \left(\mathcal{M}_m(\mathbb{T}), E_{rows,\|\cdot\|}, 0\right) \end{gathered}$$

all characterize $\mathcal{H}_m(\mathbb{T})$. (see Definition 3.1.2 on page 19.)

Furthermore, if $\mathcal{H}_m(\mathbb{T}) \neq \emptyset$, we can remove the last components of the tuple. (e.g. (O_m, E_{dev}) characterizes $\mathcal{H}_m(\mathbb{T})$, etc...)

This is in fact enough because, having characterized $\mathcal{H}_m(\mathbb{T})$, we get even more characterizations.

Proposition 4.4.3. *If $g : \mathbb{R} \to \mathbb{R}$ is strictly monotone (i.e. $x_1 < x_2 \implies g(x_1) < g(x_2)$), and if $(X, f_1, y_1), (X, f_2, y_2)$ both characterize (see Definition 3.1.2) $\mathcal{H}_m(\mathbb{T})$ with $X \subset \mathcal{M}_m(\mathbb{C})$ and $\alpha, \beta > 0$, then the following also characterize $\mathcal{H}_m(\mathbb{T})$*

$$\begin{gathered} (X, g \circ f_1, g(y_1)) \\ (X, \alpha f_1 + \beta f_2, \alpha y_1 + \beta y_2) \\ (X, E, y_1) \end{gathered}$$

where $E(M) = f_1(M^\star)$.

We also get characterizations for $\mathcal{H}_m(C_q)$ and in particular $\mathcal{H}_m$ by the following.

Proposition 4.4.4. *Letting $W \subset X$, if (X, f, y) characterizes $Y \subset X$, and $Y' = Y \cap W$, then $(W, f|_W, y)$ characterizes Y'.*

Remark 4.4.5. There are many other tuples $(O_m, \dots, \dots)$ which characterize $\mathcal{H}_m(\mathbb{T})$, and some are discussed in [5]. Corresponding algorithms can be developed, but a little experimentation revealed $(O_m, \dots, \dots)$ not to work as well as $(\mathcal{M}_m(\mathbb{T}), \dots, \dots)$ to find Hadamard matrices. We have therefore not elaborated in this direction. We note this as a subject for possible future research. In particular, is it possible to combine these formulations to search for Hadamard matrices more efficiently?

Therefore, we restrict ourselves to $\mathcal{M}_m(\pm)$. In the future, whenever we mention the "distance between elements of $\mathcal{M}_m(\pm)$," we will be referring to the Hamming distance $d(A, B) = \left|\left\{(i,j) \in \{1, ..., m\}^2 \,\middle|\, [A]_{ij} \neq [B]_{ij}\right\}\right|$. For more information on the Hamming distance, see [13].

Remark 4.4.6. In case the reader is wondering why we haven't bothered with Exhaustive Search, with $\mathcal{M}_m(\pm)$ being finite, the answer is simple. Viewing matrices as functions from indices to components, we see that $|\mathcal{M}_m(\pm)| = 2^{m^2}$, which grows extremely fast. Indeed, for $m = 32$, it is already $> 10^{307}$, which is quite impossible to Exhaustively Search naively (i.e. by visiting all elements).

Next, we describe the main algorithms used to perform optimization. We give them for finding general "complex Hadamard matrices," so that the community of researchers working on those can benefit from these algorithms, but for the rest of this thesis, we shall limit ourselves to the special case of real Hadamard matrices.

Chapter 5

Algorithms for complex Hadamard matrices

All of our algorithms use an encoding to represent matrices. When working in $\mathcal{M}_m(C_q)$ to find Butson-type q Hadamard matrices, this encoding has the advantage of being exact and requires less space and less arithmetic. When working in $\mathcal{M}_m(\mathbb{T})$ to find complex Hadamard matrices, it simplifies the computation of the gradient, allows to strictly enforce the constraint that elements be in $\mathbb{T}$, and takes about half the space. When working in $\mathcal{M}_m(\pm)$ to find real Hadamard matrices, it allows to store the matrices with a density of 1 bit per matrix element, approximately, and though we shall not explicitly discuss bitwise implementation of modular arithmetic, the encoding allows for very efficient use of the hardware. We now discuss this encoding.

Definition 5.0.1. For any $q \in \mathbb{Z}$, $\mathbb{Z}/q\mathbb{Z}$ will denote the quotient of $\mathbb{Z}$ under the relation $x \equiv y \iff x - y \in q\mathbb{Z}$, where $q\mathbb{Z} = \{qz \,|z \in \mathbb{Z}\}$.

Similarly for any $r \in \mathbb{R}$, $\mathbb{R}/r\mathbb{Z}$ will denote the quotient of $\mathbb{R}$ under the relation $x \equiv y \iff x - y \in r\mathbb{Z}$.

Remark 5.0.2. Obviously, both of these are groups under addition. On a computer, we can represent these by, respectively $[0, q) \cap \mathbb{Z}$ and $[0, r) \cap \mathbb{R} = [0, r)$. In what follows, if a variable has type $\mathbb{Z}/q\mathbb{Z}$ or $\mathbb{R}/r\mathbb{Z}$, the addition shall denote *modular* addition,

subtraction shall denote *modular* subtraction.

It is also possible to encode matrices in $\mathcal{M}_m(\mathbb{T})$ with matrices in $\mathcal{M}_m(\mathbb{R}/2\pi\mathbb{Z})$ and encode $\mathcal{M}_m(C_q)$ with matrices in $\mathcal{M}_m(\mathbb{Z}/q\mathbb{Z})$.

Here is how:

Definition 5.0.3. We define the encoding functions $\Phi_{q,pointwise} : C_q \to \mathbb{Z}/q\mathbb{Z}$, and $\Phi_{2\pi,pointwise} : \mathbb{T} \to \mathbb{R}/2\pi\mathbb{Z}$ by

$$\begin{aligned} \Phi_{q,pointwise}\left(e^{2\pi i\frac{n}{q}}\right) &= n \mod q \\ \Phi_{2\pi,pointwise}\left(e^{i\theta}\right) &= \theta \mod 2\pi \end{aligned}$$

These functions are well defined and invertible, and their inverses are

$$\begin{aligned} \Phi^{-1}_{q,pointwise}(n) &= e^{2\pi i\frac{n}{q}} \\ \Phi^{-1}_{2\pi,pointwise}(\theta) &= e^{i\theta} \end{aligned}$$

Thus, we define encoding functions on matrices $\Phi_q : \mathcal{M}_m(C_q) \to \mathcal{M}_m(\mathbb{Z}/q\mathbb{Z})$, and $\Phi_{2\pi} : \mathcal{M}_m(\mathbb{T}) \to \mathcal{M}_m(\mathbb{R}/2\pi\mathbb{Z})$ by

$$\begin{aligned} [\Phi_q(M)]_{ij} &= \Phi_{q,pointwise}\left([M]_{ij}\right) \\ [\Phi_{2\pi}(M)]_{ij} &= \Phi_{2\pi,pointwise}\left([M]_{ij}\right) \end{aligned}$$

for every i, j.

Remark 5.0.4. If M is a complex Hadamard matrix, then $\Phi_{2\pi}(M)$ is a so-called log-Hadamard matrix, and this is why we have chosen "Φ" to denote the encoding function [46, p. 137].

5.1 Algorithms

Here are the algorithms we use to find complex, Butson-type q, and real Hadamard matrices. They culminate into 6 optimization algorithms (FS[ζ][η]) which implement

the Generalized Metropolis-Hastings methods seen before as well as the Gradient Descent Methods for E_{rows} and E_{det}. These heavy-lifters are constituted of smaller pieces. The behavior of the smaller pieces are given in the algorithm's description itself, but the behavior of the heavy lifters is more intricate and is given separately. The proofs that the actual behaviors correspond to the ones in their description are presented only when something new, hard, or interesting is needed. Most proofs follow directly from the results of Appendix B and C.

After analyzing the behavior of our algorithms, we analyse their performance. For example, in Theorem 5.1.6, we show that algorithm FS[Det][Neighbor], which computes m^2 determinants of $(m \times m)$-matrices per step, can do so with only $\mathcal{O}(m^3)$ operations. Note that even for small values of m (for example, as soon as $m \geq 4$), the performance of FS[Det][Neighbor] is already empirically significantly better than the corresponding methods which would compute determinants with off-the-shelf programs such as LAPACK.

It is possible to increase the speed of FS[Det][Neighbor] at the expanse of numerical stability. We provide a family of algorithms indexed by a parameter which controls the trade-off between speed and stability, and discuss how numerical errors propagate as a function of this parameter.

Many of the ideas we have used are not new to us. Rather, we have gathered different ideas, with some well-known ones, and combined them into the FS[ζ][η] algorithms. Our goals were the following.

1. The best practical performance on a Graphical Processing Unit such as the GTX960,
2. Good work asymptotics,
3. Good linearly accessible parallelism, and
4. Good stability.

Our progress on those fronts was therefore enabled by previous work which we shall now credit.

Algorithms E[Det], U[Det][Single], U[Det][Col], and U[Det][Row] all come from the Matrix Determinant Lemma and the Sherman-Morrison Formula, described in Appendix B. Furthermore, S[Det] uses the so-called "LU decomposition with partial pivoting," which is a well-known stable method for computing the inverse and the determinant of a matrix. Programs to do this can be found, for instance, in the LAPACK library (for CPU), as well as the cuBLAS library (for GPU).

We originally introduced the encoding Φ (see Table 5.2) to save storage space, but by generalizing the algorithms, it became apparent that this encoding had some theoretical significance. Then, when reading on complex Hadamard matrices, we saw the concept of a log-Hadamard matrix, we renamed the encoding with the letter Φ. We shall not dwell too much on what made us decide to define the algorithms the way we did, but it is worth noting that the reason Φ is good for the implementation actually *changes* when we go from real Hadamard, to Butson-type q Hadamard, to complex Hadamard.

The algorithms concerning E_{rows} are original in the sense that E_{rows} was defined in the context of this thesis and all the details have been worked out from scratch, but because of the elementary nature of the definition, the author would be surprised if some precursor to these algorithms *did not* already exist under different names in the literature (though, the author hasn't found any with enough similarity to reference).

Algorithms SP[Rows], G[Rows], and SP[Det], G[Det] all come from our ability to compute the derivatives of the partial functions of $E_{rows} \circ \Phi^{-1}$ and $E_{det^2} \circ \Phi^{-1}$. There exists a formula to compute the derivative of the determinant of a *real-valued* matrix, called Jacobi's formula (see for instance [33, Part Three, Section 8.3]), but we haven't used it here. We note that $E_{det^2} \circ \Phi^{-1}$ can be obtained from the determinant by precomposing by Φ^{-1} and post-composing by $z \mapsto -z\overline{z}$, which accounts for the discrepancies. We instead used the Matrix Determinant Lemma as a starting point, and so this deserves most of the credit. Formulas for the derivatives of the partial functions are then obtained by relatively straightforward, but slightly tedious work (see Appendix C).

We also note that the algorithms are applicable in the search of both Butson-type q Hadamard matrices and of complex Hadamard matrices, but the details are slightly

Butson	$q \in \mathbb{N} \setminus \{0\}$	$\mathbb{Z}$	$\mathbb{Z}/q\mathbb{Z}$	$C_q = \{z \in \mathbb{C} \,\vert\, z^q = 1\}$	$\Phi_{q,pointwise}$	Φ_q	...
Complex	$q = 2\pi$	$\mathbb{R}$	$\mathbb{R}/2\pi\mathbb{Z}$	$\mathbb{T}$	$\Phi_{2\pi,pointwise}$	$\Phi_{2\pi}$	
Generic	q	$\mathbb{F}$	$\mathbb{F}_q$	$\mathbb{T}$	$\Phi : \mathbb{T} \to \mathbb{F}_q$	$\Phi : \mathcal{M}_m(\mathbb{T}) \to \mathcal{M}_m(\mathbb{F}_q)$	

Variable	Type
a, b, c	$\mathbb{N}$
f	$\mathbb{F}_q \setminus \{0\}$
i, j, k, l, n	$\mathbb{N}$
u, v	$\mathbb{C}$
A, B	$\mathcal{M}_m(\mathbb{T})$
C	$\mathbb{F}_q^m$
Coeff, Coeff2	$\mathbb{C}$
Inv, InvOld, InvNew, LU	$\mathcal{M}_m(\mathbb{C})$
...	

...	
M, M'	$\mathcal{M}_m(\mathbb{T})$
R	$\mathbb{F}_q^m$
Valid	$\{0, 1\}$
$\forall_{i,j\in\{1,\ldots,m\}}.X_{ij}$	$\mathbb{N}$
α	$\mathbb{C}$
$\beta, \gamma, \delta, \varepsilon$	$\mathbb{R}$
$\forall_{i,j\in\{1,\ldots,m\},k\in\{1,2\}}.\theta_{ijk}$	$\mathbb{R}/2\pi\mathbb{Z}$
$\forall_{i,j\in\{1,\ldots,m\}}.\Gamma_{ij}$	$\mathbb{R}$
$\forall_{i<k}.\Psi_{ik}$	$\mathbb{C}$

Above, we have an association between the precise concepts (Butson, complex) and the symbol used to describe them in this section. Below, we have symbols we use for variables in our algorithms and to which set their values belong.

Table 5.2: The notation used to express our algorithms.

different. In order to minimize repetition, we adopt a "generic notation," in the sense that the same symbol, when considered for the case of Butson-type q, will mean one thing, and when considered for the case of complex Hadamard matrices, will mean another.

In Table 5.2, we associate the precise concepts to their generic symbols, and we list variables we shall use in the algorithms below.

Definition 5.1.1. Let M, B, Ψ, β be as in Table 5.2, we say (B, Ψ, β) is *a proper triplet* of M for E_{rows}, and we write $\mathbb{P}_{rows}(M) = (B, \Psi, \beta)$, if

$$\begin{aligned} B &= \Phi(M) \\ \forall_k \forall_{i<k} \Psi_{ik} &= \sum_j [M]_{ij} \overline{[M]_{kj}} \\ \beta &= E_{rows}(M). \end{aligned}$$

Furthermore, if M is invertible and Inv is as in Table 5.2, we say (B, Inv, β) is *a proper triplet* of M for E_{det}, and we write $\mathbb{P}_{det}(M) = (B, \mathrm{Inv}, \beta)$, if

$$\begin{aligned} B &= \Phi(M) \\ \mathrm{Inv} &= M^{-1} \\ \beta &= |\det(M)| \end{aligned}$$

In the following we let M, M' be as in Table 5.2. For Algorithms E[Det], U[Det][Single], U[Det][Col], U[Det][Row], S[Det], SP[Det], and G[Det], we require also that M be invertible.

We start with two algorithms to compute the energy after a small change.

Algorithm E[Rows] (*Recompute E_{rows} incrementally*). Invocation:

$$\gamma \leftarrow \mathrm{Energy[Rows]}(B, \Psi, \beta, i, j, f)$$

Given the inputs B, Ψ, β, i, j, f such that $\mathbb{P}_{rows}(M) = (B, \Psi, \beta)$ and $\Phi(M') = \Phi(M) + fe_{ij}$, then we produce $\gamma = E_{rows}(M')$ without modifying the values of the inputs.

Furthermore, we do so with a work $\Upsilon \in \Theta(m)$, and a parallelism $S_\infty \in \Theta(1)$ which is linearly accessible. (See Definition 2.5.1.)

E1. [Initialize.] Set $\gamma \leftarrow \beta$. (After, $\gamma = E_{rows}(M)$.)

E2. [Remove old] Set $\gamma \leftarrow \gamma - \sum_{k<i} |\Psi_{ki}|^2 - \sum_{k>i} |\Psi_{ik}|^2$. (After, $\gamma = E_{rows}(M) - \sum_{k\neq i} \left|\left[MM^T\right]_{ik}\right|^2$.)

E3. [Add new] Set

$$\begin{aligned} \gamma \;\leftarrow\; & \gamma + \sum_{k<i} \left|\Psi_{ki} - \Phi^{-1}\left([B]_{kj} - [B]_{ij}\right) + \Phi^{-1}\left([B]_{kj} - [B]_{ij} - f\right)\right|^2 \\ + \; & \sum_{k>i} \left|\Psi_{ik} - \Phi^{-1}\left([B]_{ij} - [B]_{kj}\right) + \Phi^{-1}\left([B]_{ij} - [B]_{kj} + f\right)\right|^2 \end{aligned}$$

(After, $\gamma = E_{rows}(M) - \sum_{k\neq i} \left|\left[MM^T\right]_{ik}\right|^2 + \sum_{k\neq i} \left|\left[M'M'^T\right]_{ik}\right|^2$.) ∎

Algorithm E[Det] (*Recompute* E_{det} *incrementally*). Invocation:

$$\gamma \leftarrow \text{Energy[Det]}(B, \text{Inv}, \beta, i, j, f)$$

Given the inputs, if $\mathbb{P}_{det}(M) = (B, \text{Inv}, \beta)$ and if $\Phi(M') = \Phi(M) + fe_{ij}$, then we produce $\gamma = E_{det}(M')$ without modifying the values of the inputs.

Furthermore, we do so with a work $\Upsilon \in \Theta(1)$, and a parallelism $S_\infty \in \Theta(1)$ which is linearly accessible. (See Definition 2.5.1.)

E1. [Use Proposition B.0.3.] Set

$$\gamma \leftarrow -\left|1 + \left(\Phi^{-1}\left([B]_{ij} + f\right) - \Phi^{-1}\left([B]_{ij}\right)\right)[\text{Inv}]_{ji}\right|\beta$$

∎

Then we have four algorithms to update the proper triplets (so that they remain proper according to Definition 5.1.1).

Algorithm U[Rows][Single] (*Update* $\mathbb{P}_{rows}$ *incrementally*). Invocation:

$$\text{Update[Rows][Single]}(B, \Psi, \beta, i, j, f)$$

Given the inputs, if $\mathbb{P}_{rows}(M) = \left(B_{[in]}, \Psi_{[in]}, \beta_{[in]}\right)$ and $\Phi(M') = \Phi(M) + fe_{ij}$, then we modify these variables so as to ensure that $\mathbb{P}_{rows}(M') = \left(B_{[out]}, \Psi_{[out]}, \beta_{[out]}\right)$.

Furthermore, we do so with a work $\Upsilon \in \Theta(m)$, and a parallelism $S_\infty \in \Theta(1)$ which is linearly accessible.

U1. [Remove old.] Set $\beta \leftarrow \beta - \sum_{k<i}|\Psi_{ki}|^2 - \sum_{k>i}|\Psi_{ik}|^2$.

U2. [Update Ψ.] $[\![\forall_{k<i}]\!]$, set

$$\Psi_{ki} \leftarrow \Psi_{ki} - \Phi^{-1}\left([B]_{kj} - [B]_{ij}\right) + \Phi^{-1}\left([B]_{kj} - [B]_{ij} - f\right)$$

and $[\![\forall_{k>i}]\!]$, set

$$\Psi_{ik} \leftarrow \Psi_{ik} - \Phi^{-1}\left([B]_{ij} - [B]_{kj}\right) + \Phi^{-1}\left([B]_{ij} - [B]_{kj} + f\right)$$

U3. [Add new] Set $\beta \leftarrow \beta + \sum_{k<i} |\Psi_{ki}|^2 + \sum_{k>i} |\Psi_{ik}|^2$.

U4. [Update B.] Set $[B]_{ij} \leftarrow [B]_{ij} + f$. ∎

Algorithm U[Det][Single] (*Update* $\mathbb{P}_{det}$ *incrementally*). Invocation:

$$\text{Valid} \leftarrow \text{Update[Det][Single]}(B, \text{Inv}, \beta, i, j, f, \varepsilon)$$

Given the inputs, if $\mathbb{P}_{det}(M) = \left(B_{[in]}, \text{Inv}_{[in]}, \beta_{[in]}\right)$ and $\Phi(M') = \Phi(M) + fe_{ij}$, then if $\left|1 + \left([M']_{ij} - [M]_{ij}\right)[M^{-1}]_{ji}\right| < \varepsilon$, we will modify the inputs so as to produce Valid $= 0$ and $\mathbb{P}_{det}(M) = \left(B_{[out]}, \text{Inv}_{[out]}, \beta_{[out]}\right)$. Otherwise, we will modify to produce Valid $= 1$ and $\mathbb{P}_{det}(M') = \left(B_{[out]}, \text{Inv}_{[out]}, \beta_{[out]}\right)$.

Furthermore, we do so with a work $\Upsilon \in \Theta(m^2)$, and a parallelism $S_\infty \in \Theta(m^2)$ which is linearly accessible.

U1. [Use Proposition B.0.3.] Set

$$\text{Coeff} \leftarrow 1 + \left(\Phi^{-1}\left([B]_{ij} + f\right) - \Phi^{-1}\left([B]_{ij}\right)\right)[\text{Inv}]_{ji}$$

U2. [Well-Conditioned?] If $|\text{Coeff}| < \varepsilon$, then set Valid $\leftarrow 0$ and terminate. Otherwise, set Valid $\leftarrow 1$. (This makes sure that Proposition B.0.3 can be applied, i.e. that Coeff $\neq 0$. We would invalidate otherwise.)

U3. [Update β.] Set $\beta \leftarrow |\text{Coeff}|\,\beta$.

U4. [Use Proposition B.0.3.] Set $\text{Coeff2} \leftarrow \frac{-\left(\Phi^{-1}\left([B]_{ij}+f\right)-\Phi^{-1}\left([B]_{ij}\right)\right)}{\text{Coeff}}$.

U5. [Make updated Inv.] $[\![\forall_{k,l}]\!]$, set $[\text{InvNew}]_{kl} \leftarrow [\text{Inv}]_{kl} + \text{Coeff2}\,[\text{Inv}]_{ki}\,[\text{Inv}]_{jl}$.

U6. [Copy back.] $[\![\forall_{k,l}]\!]$, set $[\text{Inv}]_{kl} \leftarrow [\text{InvNew}]_{kl}$.

U7. [Update B.] Set $[B]_{ij} \leftarrow [B]_{ij} + f$. ∎

Algorithm U[Det][Col] (*Update* $\mathbb{P}_{det}$ *after column change*). Invocation:

$$\text{Valid} \leftarrow \text{Update[Det][Col]}(B, \text{Inv}, \beta, j, C, \varepsilon)$$

Given the inputs, if $\mathbb{P}_{det}(M) = \left(B_{[in]}, \text{Inv}_{[in]}, \beta_{[in]}\right)$ and $\Phi(M') = \Phi(M) + \sum_i C_i e_{ij}$, then if $\left|1 + \sum_i \left([M']_{ij} - [M]_{ij}\right)[M^{-1}]_{ji}\right| < \varepsilon$, we will modify so as to produce Valid $=$

0 and $\mathbb{P}_{det}(M) = \left(B_{[out]}, \mathrm{Inv}_{[out]}, \beta_{[out]}\right)$. Otherwise, Valid = 1 and $\mathbb{P}_{det}(M') = \left(B_{[out]}, \mathrm{Inv}_{[out]}, \beta_{[out]}\right)$.

Furthermore, we do so with a work $\Upsilon \in \Theta(m^2)$, and a parallelism $S_\infty \in \Theta(m)$ which is linearly accessible.

U1. [Lemma B.0.4.] $[\![\forall_i]\!]$, set $u_i \leftarrow \Phi^{-1}\left([B]_{ij}\right)\left(\Phi^{-1}(C_i) - 1\right)$.

U2. [Prop B.0.3.] $[\![\forall_k]\!]$, set $\mathrm{Coeff}_k \leftarrow \sum_{i=1}^{m} u_i [\mathrm{Inv}]_{ki}$. (Note how Coeff is not quite as it was in previous algorithm, this is because we use it at two places.)

U3. [Well-Conditioned?] If $|1 + \mathrm{Coeff}_j| < \varepsilon$, then set Valid $\leftarrow 0$, and terminate. Otherwise, set Valid $\leftarrow 1$.

U4. [Update β.] Set $\beta \leftarrow |1 + \mathrm{Coeff}_j|\,\beta$.

U5. [Prop B.0.3.] Set $\mathrm{Coeff2} \leftarrow \frac{1}{1+\mathrm{Coeff}_j}$, and then, $[\![\forall_k]\!]$, set $\mathrm{Coeff}_k \leftarrow \mathrm{Coeff2} \times \mathrm{Coeff}_k$.

U6. [Make updated Inv.] $[\![\forall_{k,l}]\!]$, set $[\mathrm{InvNew}]_{kl} \leftarrow [\mathrm{Inv}]_{kl} - \mathrm{Coeff}_k [\mathrm{Inv}]_{jl}$.

U7. [Copy back.] $[\![\forall_{k,l}]\!]$, set $[\mathrm{Inv}]_{kl} \leftarrow [\mathrm{InvNew}]_{kl}$.

U8. [Update B.] $[\![\forall_i]\!]$, set $[B]_{ij} \leftarrow [B]_{ij} + C_i$ ∎

Algorithm U[Det][Row] (*Update* $\mathbb{P}_{det}$ *after row change*). Invocation:

$$\mathrm{Valid} \leftarrow \mathrm{Update[Det][Row]}(B, \mathrm{Inv}, \beta, i, R, \varepsilon)$$

Given the inputs, similar to previous, if $\mathbb{P}_{det}(M) = \left(B_{[in]}, \mathrm{Inv}_{[in]}, \beta_{[in]}\right)$ and $\Phi(M') = \Phi(M) + \sum_j R_j e_{ij}$, then if $\left|1 + \sum_j \left([M']_{ij} - [M]_{ij}\right)[M^{-1}]_{ji}\right| < \varepsilon$, we will produce Valid $= 0$ and $\mathbb{P}_{det}(M) = \left(B_{[out]}, \mathrm{Inv}_{[out]}, \beta_{[out]}\right)$. Otherwise, we will produce Valid $= 1$ and $\mathbb{P}_{det}(M') = \left(B_{[out]}, \mathrm{Inv}_{[out]}, \beta_{[out]}\right)$.

Furthermore, we do so with a work $\Upsilon \in \Theta(m^2)$, and a parallelism $S_\infty \in \Theta(m)$ which is linearly accessible.

U1. [Lemma B.0.4.] $[\![\forall_j]\!]$, set $v_j \leftarrow \Phi^{-1}\left([B]_{ij}\right)\left(\Phi^{-1}(R_j) - 1\right)$. (After, we have $v_j = [M']_{ij} - [M]_{ij}$ for all j.)

U2. [Prop B.0.3.] $[\![\forall_l]\!]$, set $\mathrm{Coeff}_l \leftarrow \sum_{j=1}^{m} v_j [\mathrm{Inv}]_{jl}$. (After, we have $\mathrm{Coeff}_l = \sum_{j=1}^{m}\left([M']_{ij} - [M]_{ij}\right)[M^{-1}]_{jl}$ for all l.)

U3. [Well-Conditioned?] If $|1+\text{Coeff}_i| < \varepsilon$, then set Valid $\leftarrow 0$, and terminate. Otherwise, set Valid $\leftarrow 1$.

U4. [Update β.] Set $\beta \leftarrow |1+\text{Coeff}_i|\,\beta$. (After, $\beta = |\det(M')|$.)

U5. [Prop B.0.3.] Set Coeff2 $\leftarrow \frac{1}{1+\text{Coeff}_i}$. Then, $[\![\forall_l]\!]$, set $\text{Coeff}_l \leftarrow \text{Coeff2} \times \text{Coeff}_l$. (After,

$$\text{Coeff2} = \frac{1}{1+\sum_j \left([M']_{ij} - [M]_{ij}\right)[M^{-1}]_{ji}}$$

and thus $\text{Coeff}_l = \frac{\sum_{j=1}^{m}\left([M']_{ij}-[M]_{ij}\right)\left[M^{-1}\right]_{jl}}{1+\sum_j\left([M']_{ij}-[M]_{ij}\right)[M^{-1}]_{ji}}$ for every l.)

U6. [Make updated Inv.] $[\![\forall_{k,l}]\!]$, set $[\text{InvNew}]_{kl} \leftarrow [\text{Inv}]_{kl} - \text{Coeff}_l\,[\text{Inv}]_{ki}$. (After, $[\text{InvNew}] = [M'^{-1}]$.)

U7. [Copy back.] $[\![\forall_{k,l}]\!]$, set $[\text{Inv}]_{kl} \leftarrow [\text{InvNew}]_{kl}$. (After, $[\text{Inv}] = [M'^{-1}]$.)

U8. [Update B.] $[\![\forall_j]\!]$, set $[B]_{ij} \leftarrow [B]_{ij} + R_j$. (After, $B = \Phi(M) + \sum_j R_j e_{ij}$.) ∎

Now come two algorithms to compute the proper triplet (recall Definition 5.1.1) directly (and slowly) using stable methods. By not using any previous values of the proper triplet (aside from B), inserting a call to one of these algorithms will stop the propagation of errors inherent in the previous updating algorithms. When we compare our algorithms with the "naive methods," we refer to using a stable computation of the proper triplet every time we need to compute the energy of a candidate. It isn't so much naive as it is a general method that doesn't exploit the specific nature of what we are after.

Algorithm S[Rows] (*Compute $\mathbb{P}_{rows}$ with stability*). Invocation:

$$\beta, \Psi \leftarrow \text{Stable[Rows]}(B)$$

Given B with $\Phi(M) = B$, we compute the proper tuple $\mathbb{P}_{rows}(M) = (B, \Psi, \beta)$. (In fact, B does not change; we simply complete the tuple by computing Ψ and β corresponding to B.)

Furthermore, we do so with a work $\Upsilon \in \Theta(m^3)$, and a parallelism $S_\infty \in \Theta(m^2)$ which is linearly accessible.

S1. [Compute Ψ.] $[\![\forall_k \forall_{i<k}]\!]$, set $\Psi_{ik} \leftarrow \sum_j \Phi^{-1}\left([B]_{ij} - [B]_{ik}\right)$.

S2. [Compute β.] Set, $[\![\forall_k]\!]$, $\delta_k \leftarrow \sum_{i<k} |\Psi_{ik}|^2$. Then set $\beta \leftarrow \sum_k \delta_k$. ∎

Algorithm S[Det] (*Compute $\mathbb{P}_{det}$ with stability*). Invocation:

$$\beta, \mathrm{Inv} \leftarrow \mathrm{Stable[Det]}(B)$$

Given B, letting $\Phi(M) = B$, we compute the proper tuple $\mathbb{P}_{det}(M) = (B, \mathrm{Inv}, \beta)$. (Again, B doesn't change; we simply complete the tuple.)

Furthermore, we do so with a work $\Upsilon \in \Theta(m^3)$, and a parallelism $S_\infty \in \Theta(1)$ which is linearly accessible.

S1. [Compute M.] $[\![\forall_{i,j}]\!]$, set $[M]_{ij} \leftarrow \Phi^{-1}\left([B]_{ij}\right)$.

S2. [LU decomp. with Partial Pivoting.] Call LU $\leftarrow$ LUDecompose(M).

S3. [LU Determinant.] Call $\beta \leftarrow$ |LUDet(LU)|.

S4. [LU Inverse.] Call Inv $\leftarrow$ LUInv(LU). ∎

Then, we show two algorithms to compute all the partial stationary points, which seem valuable candidates as all the global minima for the partial functions must occur on such points. However, we must emphasize that these global minima aren't the same as the global minima of the energy itself. If nothing else, these algorithms showcase the fact that we were able to find closed formulas for the partial derivatives.

Algorithm SP[Rows] (*Partial Stationary Points of $E_{rows} \circ \Phi^{-1}$*). Invocation:

$$\theta \leftarrow \mathrm{StationaryPoints[Rows]}(B, \Psi)$$

Given B and Ψ such that $\Phi(M) = B$ and $\forall_k \forall_{i<k} \Psi_{ik} = \sum_j [M]_{ij} \overline{[M]_{kj}}$, then we produce θ, without modifying the inputs, such that, for every i, j we have that the set $\{\theta_{ij1}, \theta_{ij2}\}$ is, modulo 2π, equal to $\mathrm{stat}_{\mathrm{part}_{E_{rows} \circ \Phi^{-1}, B, ij}}$the set of stationary points of the partial function for the (i,j)-th component of B.

Furthermore, we do so with a work $\Upsilon \in \Theta(m^3)$, and a parallelism $S_\infty \in \Theta(m^2)$ which is linearly accessible.

SP1. [Compute complex.] $[\![\forall_{i,j}]\!]$, set

$$\alpha_{ij} \leftarrow \sum_{k>i} \left(\Phi^{-1}\left([B]_{ij} - [B]_{kj} \right) \overline{\Psi_{ik}} - 1 \right) + \sum_{k<i} \left(\Phi^{-1}\left([B]_{ij} - [B]_{kj} \right) \Psi_{ki} - 1 \right)$$

SP2. [First angle.] $[\![\forall_{i,j}]\!]$, set $\theta_{ij1} \leftarrow -\text{phase}\left(\alpha_{ij}\right)$.

SP3. [Second angle.] $[\![\forall_{i,j}]\!]$, set $\theta_{ij2} \leftarrow \theta_{ij1} + \pi$. ∎

Algorithm SP[Det] (*Partial Stationary Points of* $E_{det^2} \circ \Phi^{-1}$). Invocation:

$$\theta \leftarrow \text{StationaryPoints[Det]}(B, \text{Inv})$$

Given B, Inv such that $\Phi(M) = B$ and $\text{Inv} = M^{-1}$, then, as above, we produce θ, without modifying the inputs, such that, for every i, j we have that the set $\{\theta_{ij1}, \theta_{ij2}\}$ is, modulo 2π, equal to $\text{stat}_{\text{part}_{E_{det^2} \circ \Phi^{-1}, B, ij}}$ the set of stationary points of the partial function for the (i,j)-th component of B.

Furthermore, we do so with a work $\Upsilon \in \Theta(m^2)$, and a parallelism $S_\infty \in \Theta(m^2)$ which is linearly accessible.

SP1. [Compute complex.] $[\![\forall_{i,j}]\!]$, set $\alpha_{ij} \leftarrow [\text{Inv}]_{ji} - \left|[\text{Inv}]_{ji}\right|^2 \Phi^{-1}\left(-[B]_{ij}\right)$.

SP2. [First angle.] $[\![\forall_{i,j}]\!]$, set $\theta_{ij1} \leftarrow -[B]_{ij} - \text{phase}\left(\alpha_{ij}\right)$.

SP3. [Second angle.] $[\![\forall_{i,j}]\!]$, set $\theta_{ij2} \leftarrow \theta_{ij1} + \pi$. ∎

We now have two algorithms that compute the gradient (where the second technically computes the normalized gradient.)

Algorithm G[Rows] (*Gradient of* $E_{rows} \circ \Phi^{-1}$). Invocation:

$$\Gamma \leftarrow \text{Gradient[Rows]}(B, \Psi)$$

Given B and Ψ such that $\Phi(M) = B$ and $\forall_k \forall_{i<k} \Psi_{ik} = \sum_j [M]_{ij} \overline{[M]_{kj}}$, then we produce, without changing the inputs, $\Gamma = \nabla\left(E_{rows} \circ \Phi^{-1}\right)|_B$.

Furthermore, we do so with a work $\Upsilon \in \Theta(m^3)$, and a parallelism $S_\infty \in \Theta(m^2)$ which is linearly accessible. (Note that $\Im(a + ib) = b$ will denote the imaginary part.)

G1. [Use Lemma C.0.3.] $[\![\forall_{i,j}]\!]$, set

$$\Gamma_{ij} \leftarrow -2\left[\sum_{k>i} \Im\left(\Phi^{-1}\left([B]_{ij} - [B]_{kj}\right)\overline{\Psi_{ik}}\right) + \sum_{k<i} \Im\left(\Phi^{-1}\left([B]_{ij} - [B]_{kj}\right)\Psi_{ki}\right)\right]$$

∎

Algorithm G[Det] (*Gradient of $E_{det^2} \circ \Phi^{-1}$, divided by $|E_{det^2} \circ \Phi^{-1}|$*). Invocation:

$$\Gamma \leftarrow \text{Gradient[Det]}(B, \text{Inv})$$

Given B and Inv such that $\Phi(M) = B$ and $\text{Inv} = M^{-1}$, then we produce, without changing the inputs, $\Gamma = \frac{\nabla\left(E_{det^2} \circ \Phi^{-1}\right)|_B}{\left|E_{det^2} \circ \Phi^{-1}(B)\right|}$.

Furthermore, we do so with a work $\Upsilon \in \Theta(m^2)$, and a parallelism $S_\infty \in \Theta(m^2)$ which is linearly accessible.

G1. [Use Lemma C.0.6.] $[\![\forall_{i,j}]\!]$, set $\Gamma_{ij} \leftarrow 2\Im\left(\Phi^{-1}\left([B]_{ij}\right)[\text{Inv}]_{ji}\right)$. ∎

We have an algorithm to make sure that the gradient descent doesn't change too much B so that the proper triplet can still be updated efficiently.

Algorithm GS (*Sparsify a square matrix*). Invocation:

$$k, l, C \leftarrow \text{GradientSparsify}(\Gamma)$$

Given Γ, we pick either the row or the column which has the greatest euclidean norm (whichever is biggest overall), and then produce $k = 0$ (if we have picked a column) or $k = 1$ (if we have picked a row) and C becomes either that row or that column, and l becomes the index of that row or column.

Furthermore, we do so with a work $\Upsilon \in \Theta(m^2)$, and a parallelism $S_\infty \in \Theta(m)$ which is linearly accessible.

Let $\mathsf{Row} = 1, \mathsf{Col} = 0$.

GS1. [Sum Rows.] Set, $[\![\forall_i]\!]$, $\delta_i \leftarrow \sum_j (\Gamma_{ij})^2$.

GS2. [Sum Cols.] Set, $[\![\forall_j]\!]$, $\delta_{m+j} \leftarrow \sum_i (\Gamma_{ij})^2$.

GS3. [Find argmax.] Set $i_{max} \leftarrow \text{argmax}_{i\in\{1,\ldots,2m\}}\delta_i$. ("argmax" is like Exhaustive Search.)

GS4. [Pick Col.] If $i_{max} > m$, then set $k \leftarrow \mathsf{Col}$, $l \leftarrow i_{max} - m$, and $[\![\forall_i]\!]$, set $C_i \leftarrow \Gamma_{il}$.

GS5. [Pick Row.] If $i_{max} \leq m$, then set $k \leftarrow \mathsf{Row}$, $l \leftarrow i_{max}$, and $[\![\forall_j]\!]$, set $C_j \leftarrow \Gamma_{lj}$.

∎

To help the reader, we have compiled in Table 5.3 a summary of the behavior of the algorithms above. For precision, refer to the explicit statements. For conciseness, refer to the table.

Proof. The proofs that the behaviors are indeed as advertised in the descriptions are a direct consequence of the definition of $\mathbb{P}_{rows}(M)$, of Lemma C.0.2, Lemma C.0.3, Proposition B.0.3, Lemma B.0.4, the definition of $\mathbb{P}_{det}(M)$ and the definition of the LU decomposition (see [48]), Lemma C.0.5, and Lemma C.0.6.

From these formulas, we simply go through each step and show one after the other that as long as the right properties are satisfied up to that step, then they will be satisfied after too. As an example, we prove explicitly the result for Algorithm U[Det][Row], by verifying the properties in parenthesis.

We start with

$$\begin{aligned} B &= \Phi(M) \\ \text{Inv} &= M^{-1} \\ \beta &= |\det(M)| \\ \Phi(M') &= \Phi(M) + \sum_j R_j e_{ij} \end{aligned}$$

We go through the steps

1. For the first step, we use

$$\begin{aligned} \Phi^{-1}\left([B]_{ij}\right)\left(\Phi^{-1}(R_j) - 1\right) &= \left(\Phi^{-1}\left([B]_{ij} + R_j\right) - \Phi^{-1}\left([B]_{ij}\right)\right) \\ &= [M']_{ij} - [M]_{ij} \end{aligned}$$

Call	Precondition	Post-condition
$\gamma \leftarrow$ Energy[Rows](B,Ψ,β,i,j,f)	$\mathbb{P}_{rows}(M) = \left(B_{[in]}, \Psi_{[in]}, \beta_{[in]}\right)$ $\Phi(M') = \Phi(M) + fe_{ij}$	$\gamma = E_{rows}(M')$ Inputs Unchanged
Update[Rows][Single]$(B, \Psi, \beta, i, j, f)$		$\mathbb{P}_{rows}(M') = \left(B_{[out]}, \Psi_{[out]}, \beta_{[out]}\right)$
$\beta,\Psi \leftarrow$ Stable[Rows](B)	$B_{[in]} = \Phi(M)$	$\mathbb{P}_{rows}(M) = \left(B_{[out]}, \Psi_{[out]}, \beta_{[out]}\right)$
$\theta \leftarrow$ StationaryPoints[Rows](B, Ψ)	$\mathbb{P}_{rows}(M) = (B, \Psi, E_{rows}(M))$	$\forall_{ij}.\text{stat}_{\text{part}_{E_{rows}\circ\Phi^{-1},B,i,j}} \equiv \{\theta_{ij1}, \theta_{ij2}\}$
$\Gamma \leftarrow$ Gradient[Rows](B,Ψ)		$\Gamma = \nabla\left(E_{rows} \circ \Phi^{-1}\right)|_B$
$\gamma \leftarrow$ Energy[Det]$(B,\text{Inv},\beta,i,j,f)$	$\mathbb{P}_{det}(M) = \left(B_{[in]}, \text{Inv}_{[in]}, \beta_{[in]}\right)$ $\Phi(M') = \Phi(M) + fe_{ij}$	$\gamma = E_{det}(M')$ Inputs Unchanged
Valid $\leftarrow$ Update[Det][Single]$(B,\text{Inv},\beta,i,j,f,\varepsilon)$		$\mathcal{C}\left[1 + \left([M']_{ij} - [M]_{ij}\right)\left[M^{-1}\right]_{ji}\right]$
Valid $\leftarrow$ Update[Det][Col]$(B,\text{Inv},\beta,j,C,\varepsilon)$	$\mathbb{P}_{det}(M) = \left(B_{[in]}, \text{Inv}_{[in]}, \beta_{[in]}\right)$ $\Phi(M') = \Phi(M) + \sum_i C_i e_{ij}$	$\mathcal{C}\left[1 + \sum_i \left([M']_{ij} - [M]_{ij}\right)\left[M^{-1}\right]_{ji}\right]$
Valid $\leftarrow$ Update[Det][Row]$(B,\text{Inv},\beta,i,R,\varepsilon)$	$\mathbb{P}_{det}(M) = \left(B_{[in]}, \text{Inv}_{[in]}, \beta_{[in]}\right)$ $\Phi(M') = \Phi(M) + \sum_j R_j e_{ij}$	$\mathcal{C}\left[1 + \sum_j \left([M']_{ij} - [M]_{ij}\right)\left[M^{-1}\right]_{ji}\right]$
$\beta, \text{Inv} \leftarrow$ Stable[Det](B)	$B = \Phi(M)$	$\mathbb{P}_{det}(M) = \left(B_{[out]}, \text{Inv}_{[out]}, \beta_{[out]}\right)$
$\theta \leftarrow$ StationaryPoints[Det](B, Inv)	$\mathbb{P}_{det}(M) = (B, \text{Inv}, \lvert\det(M)\rvert)$	$\forall_{ij}.\text{stat}_{\text{part}_{E_{det^2}\circ\Phi^{-1},B,i,j}} \equiv \{\theta_{ij1}, \theta_{ij2}\}$
$\Gamma \leftarrow$ Gradient[Det](B, Inv)		$\Gamma = \frac{\nabla\left(E_{det^2}\circ\Phi^{-1}\right)|_B}{\left\lvert E_{det^2}\circ\Phi^{-1}(B)\right\rvert}$

where $\mathcal{C}[x]$ stands for:
Either

$$\begin{aligned} |x| &< \varepsilon \\ \text{Valid} &= 0 \\ \mathbb{P}_{det}(M) &= \left(B_{[out]}, \text{Inv}_{[out]}, \beta_{[out]}\right) \end{aligned}$$

or

$$\begin{aligned} |x| &\geq \varepsilon \\ \text{Valid} &= 1 \\ \mathbb{P}_{det}(M') &= \left(B_{[out]}, \text{Inv}_{[out]}, \beta_{[out]}\right) \end{aligned}$$

Table 5.3: Semantics of the building blocks for our optimization algorithms.

2. We then use the previous value of v_j to get.

$$\sum_{j=1}^{m} v_j \left[\mathrm{Inv}\right]_{jl} = \sum_{j=1}^{m} \left(\left[M'\right]_{ij} - \left[M\right]_{ij}\right) \left[M^{-1}\right]_{jl}$$

3. The 3rd step is obvious.

4. We use Proposition B.0.3. and the fact that β was equal to $|\det(M)|$ before.

5. The 5th step is obvious.

6. For every k, l, we get

$$\left[\mathrm{Inv}\right]_{kl} - \mathrm{Coeff}_l \left[\mathrm{Inv}\right]_{ki} = \left[M^{-1}\right]_{kl} - \frac{\sum_{j=1}^{m} \left(\left[M'\right]_{ij} - \left[M\right]_{ij}\right) \left[M^{-1}\right]_{jl} \left[M^{-1}\right]_{ki}}{1 + \sum_j \left(\left[M'\right]_{ij} - \left[M\right]_{ij}\right) \left[M^{-1}\right]_{ji}}$$

   which is $\left[M'^{-1}\right]_{kl}$ by Proposition B.0.3.

7. The 7th is obvious.

8. The 8th is obvious. If we terminate at this point, we have

$$\mathbb{P}_{det}(M') = \left(B_{[out]}, \mathrm{Inv}_{[out]}, \beta_{[out]}\right)$$

Hence the result follows. □

Now we can present the intermediate-sized algorithmic pieces, namely the (potentially unstable) computations of a transition procedure which satisfy the *Fair Multi-Metropolis semantics, or the gradient descent semantics.*

Definition 5.1.2. We define 4 fair transition procedures, to accompany the following algorithms. Since they all share the form[1] $\tau\left[\zeta\right]\left[\eta\right] = \mathrm{Fair}_{T,\mathcal{G},E}$, we simply need to define $\mathcal{G}$ and E:

[1] See Definition 3.4.4.

1. For $[\zeta][\eta]$ =[Rows][Neighbor], we take[2]

$$\mathcal{G}(M) = \left\{\Phi^{-1}(\Phi(M) + fe_{ij}) \,|\, f \in \{1, ..., q-1\}, i, j = 1, ..., m\right\}$$

and $E = E_{rows}$.

2. For $[\zeta][\eta]$ =[Rows][Stationary], we take

$$\mathcal{G}(M) = \Phi^{-1}\left(\mathrm{Stat}_{i,j}\left(E_{rows} \circ \Phi^{-1}, \Phi(M)\right)\right)$$

and $E = E_{rows}$.

3. For $[\zeta][\eta]$ =[Det][Neighbor], we take $\mathcal{G}$ as in 1. and $E = E_{det}$.

4. For $[\zeta][\eta]$ =[Det][Stationary], we take

$$\mathcal{G}(M) = \Phi^{-1}\left(\mathrm{Stat}_{i,j}\left(E_{det^2} \circ \Phi^{-1}, \Phi(M)\right)\right)$$

and $E = E_{det}$.

Also, we use Algorithm SF (on page 38) in a manner slightly different from the one in which we have defined it. (Instead of returning a single index, it returns the many indices required to specify a candidate. We could have simply returned a flattened[3] index, but there is no need to be so pedantic.)

Remark 5.1.3. Algorithm SF was defined as taking many inputs, among which was T. For simplicity, we assume that a specific value for T is fixed so that whenever we write "$i, j, f \leftarrow \mathrm{SelectFair}(\gamma)$," we in fact mean "$i, j, f \leftarrow \mathrm{SelectFair}(\gamma, T)$" with T having the chosen value. To be entirely formal, some of the Algorithms O and FS below in fact depend on the choice of T.

[2]Here, the lack of order over the output of $\mathcal{G}$ simply means that it isn't specified. We may choose any arbitrary one.

[3]For instance, if $(i, j) \in \{1, ..., m\}^2$, a flattened index would be, for instance, $(i-1) \times m + j \in \left\{1, ..., m^2\right\}$. This is called "flattened" since all the elements in a "2-D" matrix are now seen as elements in a "1-D" vector.

Algorithm O[Rows][Grad] (*Full optimization step, using Gradient Descent*). Invocation:

$$\text{Valid} \leftarrow \text{Optimize[Rows][Grad]}(B, \Psi, \beta, \delta)$$

Given the inputs, if $\mathbb{P}_{rows}(M) = \left(B_{[in]}, \Psi_{[in]}, \beta_{[in]}\right)$, and letting

$$M' = \Phi^{-1}\left(\tau_{Descent, E_{rows} \circ \Phi^{-1}, \delta}\left(B_{[in]}\right)\right)$$

then we produce $\mathbb{P}_{rows}(M') = \left(B_{[out]}, \Psi_{[out]}, \beta_{[out]}\right)$.

Furthermore, we do so with a work $\Upsilon \in \Theta(m^3)$, and a parallelism $S_\infty \in \Theta(m^2)$ which is linearly accessible.

O1. [Gradient.] Call $\Gamma \leftarrow \text{Gradient[Rows]}(B, \Psi)$.

O2. [Step away.] $[\![\forall_{ij}]\!]$, set $[B]_{ij} \leftarrow [B]_{ij} - \delta\Gamma_{ij}$.

O3. [Update.] Call $\beta, \Psi \leftarrow \text{Stable[Rows]}(B)$.

O4. [Automatically Valid] Set $\text{Valid} \leftarrow 1$. ∎

Algorithm O[Det][Grad] (*Full optimization step, using $\frac{1}{m}$-sparsified Gradient Descent*). Invocation:

$$\text{Valid} \leftarrow \text{Optimize[Det][Grad]}(B, \text{Inv}, \beta, \varepsilon, \delta)$$

Given the inputs, if $\mathbb{P}_{det}(M) = \left(B_{[in]}, \text{Inv}_{[in]}, \beta_{[in]}\right)$, and letting

$$M' \in \Phi^{-1}\left(\tau_{Descent, E_{det^2} \circ \Phi^{-1}, \delta, \frac{1}{m}}\left(B_{[in]}\right)\right)$$

then we produce $\text{Valid} = 1$ and

$$\mathbb{P}_{det}(M') = \left(B_{[out]}, \text{Inv}_{[out]}, \beta_{[out]}\right)$$

or produce $\text{Valid} = 0$, with the inputs unchanged.

Furthermore, we do so with a work $\Upsilon \in \Theta(m^2)$, and a parallelism $S_\infty \in \Theta(m)$ which is linearly accessible.

Let $\mathsf{Row} = 1, \mathsf{Col} = 0$.

O1. [Gradient.] Call $\Gamma \leftarrow \text{Gradient[Det]}(B, \text{Inv})$.

O2. [Sparsify.] Call $k, l, C \leftarrow \text{GradientSparsify}(\Gamma)$.

O3. [Step size.] $[\![\forall_i]\!]$, set $C_i \leftarrow -\delta\beta^2 C_i$. (The β^2 serves to compensate because Γ is the Gradient over β^2.)

O4. [Choose col.] If $k = \mathsf{Col}$, call

$$\text{Valid} \leftarrow \text{Update[Det][Col]}(B, \text{Inv}, \beta, j = l, C, \varepsilon)$$

O5. [Choose row.] If $k = \mathsf{Row}$, call

$$\text{Valid} \leftarrow \text{Update[Det][Row]}(B, \text{Inv}, \beta, i = l, R = C, \varepsilon)$$

∎

Algorithm O[ζ][Neighbor] (*Full optimization step, full neighborhood*[4]). This is a generic algorithm, for $\zeta = \text{Rows}, \text{Det}$. Invocation:

$$\text{Valid} \leftarrow \text{Optimize}[\zeta][\text{Neighbor}](B, \text{Aux}, \beta, \varepsilon)$$

Given the inputs, if $\left(B_{[in]}, \text{Aux}_{[in]}, \beta_{[in]}\right)$ is a proper tuple for M, then we produce either 1) Valid $= 1$, and such that $\left(B_{[out]}, \text{Aux}_{[out]}, \beta_{[out]}\right)$ is a proper tuple for $M' = \tau\left[\zeta\right][\text{Neighbor}]\left(M\right)$, or 2) Valid $= 0$, and the inputs are unchanged.

Furthermore, we do so with a work $\Upsilon \in \Theta\left(m^3 q\right)$ [ζ=Rows], respectively $\Upsilon \in \Theta\left(m^2 q\right)$ [ζ=Det], and a parallelism $S_\infty = \Theta\left(m^2 q\right)$ [ζ=Rows], respectively $S_\infty = \Theta\left(\frac{m^2 q}{\log_2(mq)}\right)$ [ζ=Det], which is linearly accessible (for both ζ).

O1. [Generate Candidates.] $[\![\forall_{i,j\in\{1,..,m\},f\in\{1,..,q-1\}}]\!]$, set

$$\gamma_{i,j,f} \leftarrow \text{Energy}[\zeta](B, \text{Aux}, \beta, i, j, f)$$

[4] *We have given the algorithm here such that the candidates correspond to all tuples (i, j, f) where $i, j \in \{1, ..., m\}, f \in \mathbb{Z}_q \setminus \{0\}$. When q is large, there are other possibilities to try, but the algorithms are essentially the same, so we leave it to the reader to include these modifications as the need arises.*

O2. [Select.] Call $i, j, f \leftarrow \text{SelectFair}(\gamma)$.

O3. [$\zeta = \text{Rows}$.] If $\zeta = \text{Rows}$, then set Valid $\leftarrow 1$ and call

$$\text{Update[Rows][Single]}(B, \text{Aux}, \beta, i, j, f)$$

O4. [$\zeta = \text{Det}$.] If $\zeta = \text{Det}$, then call

$$\text{Valid} \leftarrow \text{Update[Det][Single]}(B, \text{Aux}, \beta, i, j, f, \varepsilon)$$

∎

Algorithm O[ζ][Stationary] (*Full optimization step, Partial Stationary Points*). This is a generic algorithm, for $\zeta = \text{Rows}, \text{Det}$. Invocation:

$$\text{Valid} \leftarrow \text{Optimize}[\zeta][\text{Stationary}](B, \text{Aux}, \beta, \varepsilon)$$

Given the inputs, if $\left(B_{[in]}, \text{Aux}_{[in]}, \beta_{[in]}\right)$ is a proper tuple for M, then we produce either 1) Valid $= 1$, and such that $\left(B_{[out]}, \text{Aux}_{[out]}, \beta_{[out]}\right)$ is a proper tuple for $M' = \tau\left[\zeta\right][\text{Stationary}]\left(M\right)$, or 2) Valid $= 0$, and the inputs are unchanged.

Furthermore, we do so with a work $\Upsilon \in \Theta\left(m^3\right)$ [ζ=Rows], respectively $\Upsilon \in \Theta\left(m^2\right)$ [ζ=Det], and a parallelism $S_\infty \in \Theta\left(m^2\right)$ [ζ=Rows], respectively $S_\infty \in \Theta\left(\frac{m^2}{\log_2(m)}\right)$ [ζ=Det], which is linearly accessible (for both ζ).

O1. [Generate Candidates.] Call

$$\theta \leftarrow \text{StationaryPoints}[\zeta](B, \text{Aux})$$

O2. [Compute Energies.] $[\![\forall_{i,j\in\{1,..,m\},k\in\{1,2\}}]\!]$, call

$$\alpha_{ijk} \leftarrow \text{Energy}[\zeta](B, \text{Aux}, \beta, f = \theta_{ijk}, i, j)$$

O3. [Select.] Call $i, j, k \leftarrow \text{SelectFair}(\alpha)$.

O4. [$\zeta = \text{Rows}$.] If $\zeta = \text{Rows}$, then set Valid $\leftarrow 1$ and call

$$\text{Update[Rows][Single]}(B, \text{Aux}, \beta, i, j, f = \theta_{ijk})$$

O5. [$\zeta = \text{Det}$.] If $\zeta = \text{Det}$, then call

$$\text{Valid} \leftarrow \text{Update[Det][Single]}(B, \text{Aux}, \beta, i, j, f = \theta_{ijk}, \varepsilon)$$

∎

Proof. All the proofs that the performances are as advertised follow from the definition of Algorithm C, the definitions of work, span, parallelism, and linearly accessible parallelism (Section 2.5). Furthermore, by doing the proofs in order, we prove the performance of every function before proving the performance of a program which calls such functions. As an example, we prove the result for O[Rows][Neighbor], assuming the results have been proven for all preceding algorithms.

We go step by step:

1. The work for this step will be, $\Upsilon\left(\text{Energy[Row]}\right) \in \Theta\left(1\right)$ times the number of calls, $m^2 q$. We get $\Upsilon \in \Theta\left(m^2 q\right)$. The span will be the time taken if $p = \infty$, and thus, Algorithm C will start simultaneously as many tasks as there are available ones. In this case,

   $$\left|\{1, .., m\} \times \{1, .., m\} \times \mathbb{Z}_q \setminus \{0\}\right| = m^2 \left(q - 1\right)$$

   Therefore, the span is the max of all individual tasks, namely $\Upsilon_\infty \in \Theta\left(1\right)$. Here, $S_\infty \in \Theta\left(m^2 q\right)$, and we see immediately that the parallelism is linearly accessible. Indeed, $S_{S_\infty} = S_\infty$.

2. Using Proposition 3.5.7, we get a work of $\Upsilon \in \Theta\left(m^2 q\right)$, a span of

   $$\Upsilon_\infty \in \Theta\left(\log_2\left(m^2 q\right)\right) = \Theta\left(2\log_2\left(mq\right)\right) = \Theta\left(\log_2\left(mq\right)\right)$$

   and a parallelism of $S_\infty \in \Theta\left(\frac{m^2 q}{\log_2(mq)}\right)$, which is linearly accessible.

3. This branch is not taken, so we skip it and it takes a work of $\Theta(1)$.

4. As should have been proven at this point, the work for the 4-th step is $\Theta(m^2)$, the span $\Theta(1)$, the parallelism $\Theta(m^2)$, and it is linearly accessible.

Therefore, the total work is

$$\Upsilon \in \Theta\left(m^2 q\right) + \Theta\left(m^2 q\right) + \Theta(1) + \Theta\left(m^2\right) = \Theta\left(m^2 q\right)$$

the total span is

$$\Upsilon_\infty \in \Theta(1) + \Theta\left(\log_2(mq)\right) + \Theta(1) + \Theta(1) = \Theta\left(\log_2(mq)\right)$$

Therefore, the parallelism is $S_\infty \in \Theta\left(\frac{m^2 q}{\log_2(mq)}\right)$. From the properties of the parallelism, we have

$$\begin{aligned} S_{S_\infty} &\in \frac{\Theta(m^2 q)}{\frac{\Theta(m^2 q)}{\Theta\left(\frac{m^2 q}{\log_2(mq)}\right)} + \frac{\Theta(m^2 q)}{\Theta\left(\frac{m^2 q}{\log_2(mq)}\right)} + \frac{\Theta(1)}{\Theta(1)} + \frac{\Theta(m^2)}{\Theta\left(\frac{m^2 q}{\log_2(mq)}\right)}} \\ &= \Theta\left(\frac{m^2 q}{\log_2(mq)}\right) \\ &\equiv S_\infty \end{aligned}$$

□

To help the reader, we have compiled in Table 5.4 a summary of the performance of the algorithms above.

And now, the culmination of all the little pieces, a stabilized version of the previous Algorithms O[ζ][η]. We present the algorithm first, and then we state its behavior, as it depends on ζ and η.

Algorithm FS[ζ][η] (*Transition procedure with Fast and Stable methods*). This is a generic procedure, where $\zeta = \text{Rows}, \text{Det}$ and $\eta = \text{Stationary}, \text{Neighbor}, \text{Grad}$. Invocation:

$$\text{Valid} \leftarrow \text{FastStable}[\zeta][\eta](n, B, \varepsilon, \delta)$$

	Υ	S_∞
E[Rows]	$\Theta(m)$	$\Theta(1)$
E[Det]	$\Theta(1)$	$\Theta(1)$
U[Rows][Single]	$\Theta(m)$	$\Theta(1)$
U[Det][Single]	$\Theta(m^2)$	$\Theta(m^2)$
U[Det][Col]	$\Theta(m^2)$	$\Theta(m)$
U[Det][Row]		
S[Rows]	$\Theta(m^3)$	$\Theta(m^2)$
S[Det]	$\Theta(m^3)$	$\Theta(1)$
SP[Rows]	$\Theta(m^3)$	$\Theta(m^2)$
SP[Det]	$\Theta(m^2)$	$\Theta(m^2)$
G[Rows]	$\Theta(m^3)$	$\Theta(m^2)$
G[Det]	$\Theta(m^2)$	$\Theta(m^2)$
GS	$\Theta(m^2)$	$\Theta(m)$
O[Rows][Grad]	$\Theta(m^3)$	$\Theta(m^2)$
O[Det][Grad]	$\Theta(m^2)$	$\Theta(m)$
O[Rows][Neighbor]	$\Theta(m^3q)$	$\Theta(m^2q)$
O[Det][Neighbor]	$\Theta(m^2q)$	$\Theta\left(\frac{m^2q}{\log_2(mq)}\right)$
O[Rows][Stationary]	$\Theta(m^3)$	$\Theta(m^2)$
O[Det][Stationary]	$\Theta(m^2)$	$\Theta\left(\frac{m^2}{\log_2 m}\right)$

All parallelisms are linearly accessible.

Table 5.4: Performance for small and intermediate sized algorithmic pieces.

FS1. [Proper tuple.] Call

$$\beta, \text{Aux} \leftarrow \text{Stable}[\zeta](B)$$

FS2. [Start Valid.] Set Valid $\leftarrow 1$.

FS3. [Loop.] If $n = 0$ or Valid $= 0$, then terminate.

FS4. [$\eta =$Grad,$\zeta =$Rows.] If $\eta =$ Grad and $\zeta =$ Rows, then call

$$\text{Valid} \leftarrow \text{Optimize}[\text{Rows}][\text{Grad}](B, \Psi, \beta, \delta)$$

FS5. [$\eta =$Grad,$\zeta =$Rows] If $\eta =$ Grad and $\zeta =$ Det, then call

$$\text{Valid} \leftarrow \text{Optimize}[\text{Det}][\text{Grad}](B, \text{Inv}, \beta, \varepsilon, \delta)$$

FS6. [Otherwise.] If $\eta \neq$ Grad, then call

$$\text{Valid} \leftarrow \text{Optimize}[\zeta][\eta](B, \text{Aux}, \beta, \varepsilon)$$

FS7. [Loop] Set $n \leftarrow n - 1$, then go back to FS3. ∎

Theorem 5.1.4. *(Exact Semantics). Consider different values for ζ and η:*

1. *If $\eta \neq$ Grad, given the inputs $B_{[in]}, n_{[in]}$, we modify B and n so as to produce either* Valid $= 1$ and

$$\Phi^{-1}\left(B_{[out]}\right) = \left(\tau\left[\zeta\right]\left[\eta\right]\right)^{n_{[in]}}\left(\Phi^{-1}\left(B_{[in]}\right)\right)$$

   or Valid $= 0$ and

$$\Phi^{-1}\left(B_{[out]}\right) = \left(\tau\left[\zeta\right]\left[\eta\right]\right)^{n_{[in]}-\left(n_{[out]}+1\right)}\left(\Phi^{-1}\left(B_{[in]}\right)\right)$$

   where $\tau\left[\zeta\right]\left[\eta\right]$ is the transition procedure described in Definition 5.1.2.

2. *If $\eta =$* Grad *and $\zeta =$* Rows*, given the inputs $B_{[in]}, n_{[in]}$, we modify B and n so*

as to produce

$$B_{[out]} \equiv \left(\tau_{Descent,E_{rows}\circ\Phi^{-1},\delta}\right)^{n_{[in]}}\left(B_{[in]}\right)$$

modulo 2π.

3. *If* $\eta = \text{Grad}$ *and* $\zeta = \text{Det}$, *given the inputs* $B_{[in]}, n_{[in]}$, *we modify* B *and* n *so as to produce either* Valid $= 1$ and

$$B_{[out]} \in \left(\tau_{Descent,E_{rows}\circ\Phi^{-1},\delta,\frac{1}{m}}\right)^{n_{[in]}}\left(B_{[in]}\right)$$

(modulo 2π*)* or Valid $= 0$ and

$$B_{[out]} \in \left(\tau_{Descent,E_{rows}\circ\Phi^{-1},\delta,\frac{1}{m}}\right)^{n_{[in]}-\left(n_{[out]}+1\right)}\left(B_{[in]}\right)$$

(modulo 2π*).*

Proof. We can stand on the shoulders of the previous results on the behavior of the smaller algorithmic pieces we have seen in this chapter, and only show a few extra things. For instance, we have that if $A_i, i = 1, ..., m$ are positive numbers and $i_{max} = \text{argmax}_i A_i$, then $\sum_i A_i \leq mA_{i_{max}}$, which is sufficient to establish that Algorithm O[Det][Grad] in fact implements an $\frac{1}{m}$-sparsified gradient descent.

Furthermore, we see the exponent $n_{[in]} - \left(n_{[out]} + 1\right)$ arises when Valid $= 0$. This is because we decrement n before checking if Valid $= 0$, in step FS7. Therefore, to get the number of times we went through the loop, we take $n_{[in]} - n_{[out]}$, and since the last time didn't modify B, we see that to get the real number we must subtract 1, hence $n_{[in]} - \left(n_{[out]} + 1\right)$.

Also note the (modulo 2π). This is simply because, the gradient step should be defined on $\mathbb{T}^{m^2}$ rather than $\mathbb{R}^{m^2}$, since the Encoding Φ is only defined up to a multiple of 2π. □

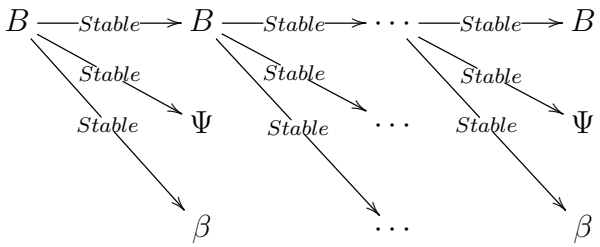
Remark 5.1.5. Assuming (potentially) inexact arithmetic in a call to Algorithm FS[ζ][η] then, any divergence between $\mathbb{P}_\zeta\left(\Phi^{-1}(B)\right)$ [the exact proper tuple corresponding to $\Phi^{-1}(B)$] and (B, Aux, β) [the actual values in the computer's memory] while the computer is at step FS3 must have been the result of a sequence of operations proceeding

as follows:

- If $(\zeta, \eta) = (\text{Rows}, \text{Grad})$, the sequence has only one element, namely a call to

$$\beta,\Psi \leftarrow \text{Stable[Rows]}(B)$$

with $B = \left[\mathbb{P}_{rows}\left(\Phi^{-1}\left(\mathcal{B}\right)\right)\right]_1$ [the exact first component of the proper tuple of $\mathcal{B}$]. If Algorithm FS[Rows][Grad] is called many times in succession, the graph of dependencies between values looks like



- For other values of $[\zeta][\eta]$,

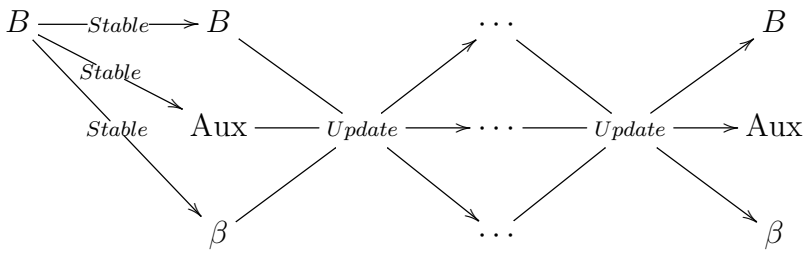
  1. We begin with $\mathcal{B}$, and $B = \left[\mathbb{P}_{\zeta}\left(\Phi^{-1}\left(\mathcal{B}\right)\right)\right]_1$,
  2. then we call $\beta,\text{Aux} \leftarrow \text{Stable}\left[\zeta\right](B)$ once,
  3. and then we call ... $\leftarrow$ Update $\left[\zeta\right]\left[\eta\right](B, \text{Aux}, \beta, ...)$ between 0 and n times, with various values substituted for the ellipsis, potentially different for every call, and each successive call takes as input the *output*

$$\left(B_{[out]}, \text{Aux}_{[out]}, \beta_{[out]}\right)$$

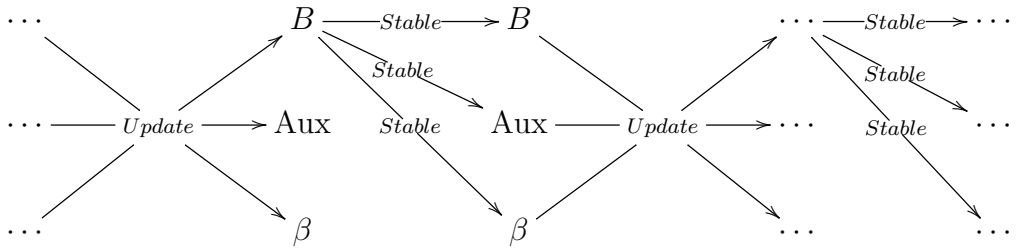
  of the previous call.

If Algorithm FS$[\zeta][\eta]$ (for values other than [Rows][Grad]) is called once, the graph

of dependencies between values looks like



If it is called multiple times, the graph of dependencies looks like



We see how the presence of a call to Stable $[\zeta]$ renders irrelevant the errors in previous values of Aux and β. Since B is always exact with respect to itself (by definition), Stable $[\zeta]$ erects a wall which accumulated errors cannot cross. By choosing n, we can control how long the chains of operations not including Stable $[\zeta]$ is allowed to be. By limiting the compounding of unstable numerical dependent operations, we in effect *stabilize* these operations.

Now we come to the performance analysis of Algorithm FS$[\zeta][\eta]$.

Theorem 5.1.6. *Provided that Algorithm FS*$[\zeta][\eta]$ *terminate with* Valid $= 1$*, it will have the following performance:*

1. *For $\zeta =$ Rows and $\eta =$ Neighbor, the work over n will be $\frac{\Upsilon}{n} \in \Theta(m^3 q)$, the parallelism, which is linearly accessible, will be $S_\infty \in \Theta(m^2 q)$. Furthermore, since the algorithm calls Algorithm $E[\zeta]$, $\#_E \in \Theta(nm^2 q)$ times, the amortized work per call to $E[\zeta]$ is $\frac{\Upsilon}{\#_E} \in \Theta(m)$. (This means that the speedup compared to the naive method is $\Theta(m^2)$.)*

2. *For ζ = Rows and η = Stationary, the work over n will be $\frac{\Upsilon}{n} \in \Theta(m^3)$, the parallelism, which is linearly accessible, will be $S_\infty \in \Theta(m^2)$. Furthermore, since the algorithm calls Algorithm $E[\zeta]$, $\#_E \in \Theta(nm^2)$ times, the amortized work per call to $E[\zeta]$ is $\frac{\Upsilon}{\#_E} \in \Theta(m)$. (This means that the speedup compared to the naive method is $\Theta(m^2)$.)*

3. *For ζ = Rows and η = Grad, the work over n will be $\frac{\Upsilon}{n} \in \Theta(m^3)$, the parallelism, which is linearly accessible, will be $S_\infty \in \Theta(m^2)$.*

4. *For ζ = Det and η = Neighbor,*

   (a) *If $n \in o\left(\frac{m}{q}\right)$, the work over n will be $\frac{\Upsilon}{n} \in \Theta\left(\frac{m^3}{n}\right)$, the parallelism, which is linearly accessible, will be $S_\infty \in \Theta(1)$. Furthermore, since the algorithm calls Algorithm $E[\zeta]$, $\#_E \in \Theta(nm^2q)$ times, the amortized work per call to $E[\zeta]$ is $\frac{\Upsilon}{\#_E} \in \Theta\left(\frac{m}{nq}\right)$. (This means that the speedup compared to the naive method is $\Theta(m^2nq)$.) [Here, the call to Algorithm $S[\zeta]$ still dominates the work.]*

   (b) *If $n \in o\left(\frac{m^3}{\log_2 m}\right)$ and $n \in \Omega\left(\frac{m}{q}\right)$, the work over n will be $\frac{\Upsilon}{n} \in \Theta(m^2q)$, the parallelism, which is linearly accessible, will be $S_\infty \in \Theta\left(\frac{nq}{m}\right)$. Furthermore, since the algorithm calls Algorithm $E[\zeta]$, $\#_E \in \Theta(nm^2q)$ times, the amortized work per call to $E[\zeta]$ is $\frac{\Upsilon}{\#_E} \in \Theta(1)$. (This means that the speedup compared to the naive method is $\Theta(m^3)$.) [Here, the calls to $E[\zeta]$ have finally overcome those to $S[\zeta]$ for the work. The parallelism is determined by the ratio of calls to $E[\zeta]$ and to $S[\zeta]$.]*

   (c) *If $n \in \Omega\left(\frac{m^3}{\log_2 m}\right)$, the work over n (as above) will be $\frac{\Upsilon}{n} \in \Theta(m^2q)$, the parallelism, which is linearly accessible, will be $S_\infty \in \Theta\left(\frac{m^2q}{\log_2 m}\right)$. Furthermore, since the algorithm calls Algorithm $E[\zeta]$, $\#_E \in \Theta(nm^2q)$ times, the amortized work per call to $E[\zeta]$ is $\frac{\Upsilon}{\#_E} \in \Theta(1)$. (This means that the speedup compared to the naive method is $\Theta(m^3)$.) [Here, the parallelism is dominated by the selection.]*

5. *For ζ = Det and η = Stationary,*

(a) *If* $n \in o(m)$, *the work over* n *will be* $\frac{\Upsilon}{n} \in \Theta\left(\frac{m^3}{n}\right)$, *the parallelism, which is linearly accessible, will be* $S_\infty \in \Theta(1)$. *Furthermore, since the algorithm calls Algorithm* $E[\zeta]$, $\#_E \in \Theta(nm^2)$ *times, the amortized work per call to* $E[\zeta]$ *is* $\frac{\Upsilon}{\#_E} \in \Theta\left(\frac{m}{n}\right)$. *(This means that the speedup compared to the naive method is* $\Theta(m^2 n)$.*) [Here, the call to Algorithm* $S[\zeta]$ *still dominates the work.]*

(b) *If* $n \in o\left(\frac{m^3}{\log_2 m}\right)$ *and* $n \in \Omega(m)$, *the work over* n *will be* $\frac{\Upsilon}{n} \in \Theta(m^2)$, *the parallelism, which is linearly accessible, will be* $S_\infty \in \Theta\left(\frac{n}{m}\right)$. *Furthermore, since the algorithm calls Algorithm* $E[\zeta]$, $\#_E \in \Theta(nm^2)$ *times, the amortized work per call to* $E[\zeta]$ *is* $\frac{\Upsilon}{\#_E} \in \Theta(1)$. *(This means that the speedup compared to the naive method is* $\Theta(m^3)$.*) [Here, the calls to* $E[\zeta]$ *have finally overcome those to* $S[\zeta]$ *for the work. The parallelism is determined by the ratio of calls to* $E[\zeta]$ *and to* $S[\zeta]$.*]*

(c) *If* $n \in \Omega\left(\frac{m^3}{\log_2 m}\right)$, *the work over* n *(as above) will be* $\frac{\Upsilon}{n} \in \Theta(m^2)$, *the parallelism, which is linearly accessible, will be* $S_\infty \in \Theta\left(\frac{m^2}{\log_2 m}\right)$. *Furthermore, since the algorithm calls Algorithm* $E[\zeta]$, $\#_E \in \Theta(nm^2)$ *times, the amortized work per call to* $E[\zeta]$ *is* $\frac{\Upsilon}{\#_E} \in \Theta(1)$. *(This means that the speedup compared to the naive method is* $\Theta(m^3)$.*) [Here, the parallelism is determined by the selection.]*

6. *For* $\zeta = Det$ *and* $\eta = Grad$,

(a) *If* $n \in o(m)$, *the work over* n *will be* $\frac{\Upsilon}{n} \in \Theta\left(\frac{m^3}{n}\right)$, *the parallelism, which is linearly accessible, will be* $S_\infty \in \Theta(1)$. *[Here, the call to Algorithm* $S[\zeta]$ *still dominates the work.]*

(b) *If* $n \in o(m^2)$ *and* $n \in \Omega(m)$, *the work over* n *will be* $\frac{\Upsilon}{n} \in \Theta(m^2)$, *the parallelism, which is linearly accessible, will be* $S_\infty \in \Theta\left(\frac{n}{m}\right)$. *[Here, the calls to* $G[\zeta]$ *and the updates have finally overcome those to* $S[\zeta]$ *for the work. The parallelism is determined by the ratios of the updates to the calls to* $S[\zeta]$.*]*

Algorithm		$\frac{\Upsilon}{n}$	S_∞	$\frac{\Upsilon}{\#_E}$
FS[Rows][Neighbor]		$\Theta(m^3q)$	$\Theta(m^2q)$	$\Theta(m)$
FS[Rows][Stationary]		$\Theta(m^3)$	$\Theta(m^2)$	$\Theta(m)$
FS[Rows][Grad]		$\Theta(m^3)$	$\Theta(m^2)$	N/A
FS[Det][Neighbor]	$n \in o\left(\frac{m}{q}\right)$	$\Theta\left(\frac{m^3}{n}\right)$	$\Theta(1)$	$\Theta\left(\frac{m}{nq}\right)$
	$n \in o\left(\frac{m^3}{\log_2 m}\right)$ and $n \in \Omega\left(\frac{m}{q}\right)$	$\Theta(m^2q)$	$\Theta\left(\frac{nq}{m}\right)$	$\Theta(1)$
	$n \in \Omega\left(\frac{m^3}{\log_2 m}\right)$	$\Theta(m^2q)$	$\Theta\left(\frac{m^2q}{\log_2 m}\right)$	$\Theta(1)$
FS[Det][Stationary]	$n \in o(m)$	$\Theta\left(\frac{m^3}{n}\right)$	$\Theta(1)$	$\Theta\left(\frac{m}{n}\right)$
	$n \in o\left(\frac{m^3}{\log_2 m}\right)$ and $n \in \Omega(m)$	$\Theta(m^2)$	$\Theta\left(\frac{n}{m}\right)$	$\Theta(1)$
	$n \in \Omega\left(\frac{m^3}{\log_2 m}\right)$	$\Theta(m^2)$	$\Theta\left(\frac{m^2}{\log_2 m}\right)$	$\Theta(1)$
FS[Det][Grad]	$n \in o(m)$	$\Theta\left(\frac{m^3}{n}\right)$	$\Theta(1)$	N/A
	$n \in o(m^2)$ and $n \in \Omega(m)$	$\Theta(m^2)$	$\Theta\left(\frac{n}{m}\right)$	
	$n \in \Omega(m^2)$	$\Theta(m^2)$	$\Theta(m)$	

Table 5.5: The performance aspects of Algorithms FS$[\zeta][\eta]$, for various ζ and η, provided that the algorithm returns Valid $= 1$.

(c) *If $n \in \Omega(m^2)$, the work over n (as above) will be $\frac{\Upsilon}{n} \in \Theta(m^2)$, the parallelism, which is linearly accessible, will be $S_\infty \in \Theta(m)$. [Here, both the work and parallelism has reached the limit imposed by the updates.]*

We have summarized the contents of the previous theorem in Table 5.5.

5.2 The Quest for Performance

After having studied the optimization literature, carried various numerical experiments, and learned that the Hadamard Conjecture had been solved for all $m < 667$, it became clear that, without a miracle, the Hadamard Conjecture would be out of reach of basic optimization techniques. But because such techniques have broader applicability than the Hadamard Conjecture, we decided to take them as far as we could. The first step was to try various algorithms, compare their convergence numbers, etc.. To do this, we broadened the usual Metropolis-Hastings method into a transition procedure with arbitrary generation, and selection. We then tried many such procedures, with many different parameters, such as thermal energy. However, we found that, relatively soon, we reached a point of diminishing returns where the added complexity of the methods couldn't be justified by their slightly smaller convergence numbers. This is why we settled on the Fair selection and the Satisficing selection (see Section 3.5), paired with some simple generation procedure, such as the one producing all nearest neighbors.

With this "semantics fixed," we turned to the task of advancing in the induced Markov chain faster. There were roughly two ways of accomplishing this:

1. Using the properties of Hadamard matrices and of $\mathcal{M}_m(\pm)$ to find more efficient ways of computing the energy and

2. Using the properties of modern computers, in order to increase the performance of the implementation.

Both of these ways of improving speed still suffer from eventual diminishing returns, so we divided our efforts between both equally. The first way has already been sufficiently chronicled in Section 5.1. Now, we briefly give a sense of the second.

Starting out, we had almost no working knowledge of how modern computers work. So we investigated parallelism, of the supercomputer variety at first, by buying a few Raspberry Pi (cheap computers), and building a network of cooperating computers. The main lesson here was that communication between computers working on the same problem is both very complicated, and impractical when the latency of

communication becomes comparable with the amount of work each computer can do without communicating with the others.

Another lesson was how intricate the performance of modern processors has become. Without a good working understanding of things like Pipelining, Cache, Single-Instruction-Multiple-Data, Out-of-order Execution, Latency vs. Bandwidth vs. Throughput, it is almost impossible to reason correctly about what program will run faster on what processor. After learning about this landscape of details and concerns, we turned to the history, or the evolution of computer architecture, to try to understand what among this torrent of details were likely to remain relevant in the near future, and what was purely ephemeral. When surveying, for instance, the performance of CPUs versus the performance of GPUs, we see that the first has more or less stagnated in the last decade, whereas the second keeps compounding orders of magnitude in performance increases.

Because of this, we decided to investigate as much as possible the performance of CPUs and GPUs. Here are a few list of the resources we made heavy use of (at least those we generally recommend as being quite useful.)

- Learning to program and understand GPUs:
  - The programming manuals, architecture manuals, machine model, libraries in the CUDA ecosystem. (CUDA is a programming language almost identical to the C programming language which allows one to directly program any GPU from NVIDIA.) See for instance [2].
  - The Stanford CUDA lectures on GPU programming. See for instance [1].
- Learning to program and understand CPUs:
  - Agner Fog's manuals on the intricacies of CPU performance. See [17] for a table of the instructions that CPUs can execute directly, together with the associated performance. The more complete page where all the manuals can be found is [16, "Software optmization ressources"].
  - The C Programming Language, see [24].

- Learning how experts think about and solve performance problems in practice:
  - Mike Acton's talk about "Data-Oriented Design." See [3].
  - Casey Muratori's "Handmade Hero" course. See [35].
- Understand computer science better as a whole, together with its connection to mathematics:
  - The Art of Computer Programming, Volumes 1 to 4A. See [26], etc..

To conclude this chapter, we simply emphasize that the algorithms are the way they are because of our quest for actual performance on hardware such as the GTX960. Of course, the mathematical properties of our algorithms do not depend on the architecture of the GTX960, but the practical performance certainly does. When it comes to parallel performance, a few things have a huge impact:

1. The time it takes to communicate information between various groups of processors.
2. The ability of two adjacent processors to do different things at the same time.
3. Amount of "fast" memory.
4. Penalties for non-bulk operations.

If we wish to have a set of processors cooperate in performing an iteration of generation-evaluation-selection, then the latency in communication between them had better be small when compared with the time between necessary communications. Since generation depends on previous selection, evaluation depends on generation, and selection depends on evaluation, communication must occur approximately three times per iteration. If we wish to do 1 million iterations per second, then the communication latency must be much smaller than 0.3 microseconds, otherwise most of the time will be spent waiting.

This phenomenon limits how many processors can cooperate on the same iteration. We expect latency in communication between large numbers of processors to remain

long, even with future generations of the NVIDIA GPUs. However, we expect the total number of processors to increase substantially, and so we expect to be able to run many different instances of our optimization algorithms on future GPUs, with each instance having a similar speed as it currently does on the GTX960.

However, if we wish to explore higher dimensions m, then the number of iterations per second will drop, and so the latency of communication will matter less, and so larger groups of processors could be brought to bear. The large amounts of parallelism inherent in the algorithms would allow many processors to cooperate once the latency isn't an issue.

Furthermore, the GTX960 has its processors in groups of 32 (warps) and within such groups, the instructions performed must be the same, with only the data allowed to vary. This means that not all algorithms with a lot of parallelisms will be efficiently implementable. This, combined with the relatively small amount of "fast" memory available, imposes many constraints.

The improvements we have seen in the asymptotical performance (e.g. going from a work of $\Theta(m^3)$ to a work of $\Theta(m^2)$) were not what guided our choice of algorithms. Instead, we used the restrictions of the GTX960, and the performance for a fixed value of m (e.g. $m = 32$). However, in the interest of communicating the merit of these new algorithms, we have decided to discuss their asymptotical performance (work, parallelism, etc.).

It is interesting to know that the advantage these algorithms hold over the more naive alternatives does extend beyond the values of m we have considered, but asymptotical improvements would not have been enough to justify our adoption of these algorithms. It is possible to get deceived by looking at asymptotic performance improvements, since this isn't a true measure of the performance for small values of m. If we design the algorithms with asymptotics as our criterion of success, it is possible to end up with good asymptotics and terrible performance for the values of m of greater interest to us.

This is why the asymptotics are probably best used to *explain* rather than to *predict* good performance. Of course, all of this heavily depends on the fact that we are dealing with small values of m. For problems where $m > 10^9$, say, then the

asymptotics probably give a more reliable measure of actual performance.

Chapter 6

(ε, δ)-Quasi-Hadamard matrices

In this chapter, we develop a theoretical understanding of how matrices far from $\mathcal{H}_m$ can still have a small value for E_{rows}. This is the concept of an (ε, δ)-Quasi-Min (Definition 3.1.10 on page 22). We pay special attention to (ε, δ)-Quasi-Hadamard matrices, namely matrices $M \in \text{Optimize}_{\mathcal{M}_m(\pm), E_{rows}, 0}[\varepsilon]$ such that $M \notin \cup_{H \in \mathcal{H}_m} \mathcal{B}_\delta(H)$. In addition to being an unexpected byproduct of the optimization methods we have developed, the existance of such matrices give hints for why finding Hadamard matrices is difficult.

With this theoretical understanding of these matrices, we devise an algorithm to find provably valid lower bounds and, in practice, tight enough to be useful, on the distance from $\mathcal{H}_m$.

Finally, we show, using the tools developed in this chapter that a matrix, which we have found with the tools developed in Chapters 3 and 5, is a "Co-Quasi-Min" for all the functions over $\mathcal{M}_m(\pm)$ we have used to characterize Hadamard matrices.

6.1 How can matrices far from $\mathcal{H}_m$ still have a small energy?

Of necessity, we begin with definitions and theory, but briefly, we sketch the phenomenon we will study. Let $M \in \mathcal{M}_m(\pm)$, and let $d = d_{Hamming}(M, \mathcal{H}_m)$. By definition, there must exist a finite sequence of matrices $H^{(i)} \in \mathcal{M}_m(\pm)$, for $0 \leq i \leq d$ where $H^{(0)} = M$, $H^{(d)} \in \mathcal{H}_m$, and $d_{Hamming}\left(H^{(i)}, H^{(i+1)}\right) = 1$. Furthermore, we will have, if $d \neq 0$, that $H^{(i)} \notin \mathcal{H}_m$ for $i < d$. For such a sequence, we can consider the symmetric products $H^{(i)}H^{(i)T}$. More specifically, we will see that the matrix $H^{(i+1)}H^{(i+1)T} - H^{(i)}H^{(i)T}$ has a regular form, determined by $H^{(i+1)} - H^{(i)}$.

By studying these changes, one might hope to recover $H^{(d)} - H^{(0)}$ from $H^{(d)}H^{(d)T} - H^{(0)}H^{(0)T}$. Since the latter is only a function of M, and the former allows us to find $H^{(d)}$ if we know M, this would be quite useful. As it turns out, however, the best we can do in practice will be to get a non-trivial lower bound on d and partial information about $H^{(d)}$.

Informally, the fact that this "encoding process" is not "perfectly decodable" is connected to the fact that it is possible to move a "large distance" away from $\mathcal{H}_m$ and still have "$MM^T \approx m\mathbb{I}$." This in turn will explain why we can find matrices far from $\mathcal{H}_m$ with a relatively small value of E_{rows}. A more precise discussion of these notions follows.

To help us study the *actual* "encoding process," we also study a similar one which is decodable, up to reordering of the changes.

Definition 6.1.1. We define a few subsets of $I = \{1, ..., m\}^2$ (the indices of $(m \times m)$-matrices):

1. The set $\text{Up} = \{(i, j) \in I \,|i < j\}$, (the upper triangle)
2. For $i \in \{1, ..., m\}$, the set $\text{Row}_i = \{(k, j) \in I \,|k = i\}$, (the i-th row)
3. For $j \in \{1, ..., m\}$, the set $\text{Col}_j = \{(i, k) \in I \,|k = j\}$, (the j-th column)
4. The set $\text{Diag} = \{(i, j) \in I \,|i = j\}$, (the diagonal)

5. For $i \in \{1, ..., m\}$, the set $✠_i = \{(j,k) \in I \,|\, j \neq k \text{ and either } i = j \text{ or } i = k\}$, (the i-th cross)

Note that the i-th cross is just the union of the i-th row and the i-th column, minus the diagonal. ($✠_i = (\text{Row}_i \cup \text{Col}_i) \setminus \text{Diag}$)

Remark 6.1.2. Let $\mathcal{A} \subseteq \{1, ..., m\}^2$. By seeing matrices in $\mathcal{M}_m(R)$ as functions in $R^{\{1,...,m\}^2}$, we can think of functions in $R^{\mathcal{A}}$ as "partial" matrices, and store them much as we stored actual matrices, in a computer.

Definition 6.1.3. For a set $\mathcal{I}$ and $\mathcal{A} \subset \mathcal{I}$, we define *the indicator function* of $\mathcal{A}$, written $\mathbf{1}_{\mathcal{A}} : \mathcal{I} \to \{0, 1\}$, as

$$\mathbf{1}_{\mathcal{A}}(I) = \begin{cases} 1 & I \in \mathcal{A} \\ 0 & I \notin \mathcal{A} \end{cases}$$

for every $I \in \mathcal{I}$.

Furthermore, we define a set of functions $(\pm\mathbf{1})_{\mathcal{A}} \subset \{-1, 0, 1\}^{\mathcal{I}}$ called the *interfered indicator functions* of $\mathcal{A}$ as

$$(\pm\mathbf{1})_{\mathcal{A}} = \{f : \mathcal{I} \to \{-1, 0, 1\} \,|\, \text{The support of } f \text{ is exactly } \mathcal{A}.\}$$

Remark 6.1.4. We see that $|(\pm\mathbf{1})_{\mathcal{A}}| = 2^{|\mathcal{A}|}$. Furthermore, $\mathbf{1}_{\mathcal{A}} \in |(\pm\mathbf{1})_{\mathcal{A}}|$.

In what follows, we shall fix $\mathcal{I} = \{1, ..., m\}^2$ and fix $\mathcal{A} = ✠_i$, for $1 \leq i \leq m$. Since $\mathbf{1}_{✠_i} \in \mathcal{M}_m(\mathbb{N})$ and $(\pm\mathbf{1})_{✠_i} \subset \mathcal{M}_m(\mathbb{Z})$, we can picture them as a matrix and a set of matrices, respectively.

Definition 6.1.5. Given two matrices $M, M' \in \mathcal{M}_m(\pm)$ we define the *flipset*

$$\text{Flipset}(M, M') = \left\{(i,j) \in \{1, ..., m\}^2 \,|\, [M]_{ij} \neq [M']_{ij}\right\}$$

In other words, the set of components we would have to flip to go from M to M'.

Remark 6.1.6. Notice that the Hamming distance between M and M' is $|\text{Flipset}(M, M')|$.

We denote by $\text{Flipset}(M)$ the set of all $\text{Flipset}(M, H)$ for $H \in \mathcal{H}_m$ such that $|\text{Flipset}(M, H)| = d(M, \mathcal{H}_m)$.

Obviously, if we could compute $\text{Flipset}(M)$ easily, we could use it to decide whether $\mathcal{H}_m \neq \emptyset$. Indeed, we could just pick any matrix in dimension m and compute $\text{Flipset}(M)$. If $\text{Flipset}(M) = \emptyset$, then $\mathcal{H}_m = \emptyset$. Otherwise, we could take the first element, and flip M appropriately to get a Hadamard matrix. However, things are not so simple.

Definition 6.1.7. We let $F \subset \{1, ..., m\}^2$, and define the *Direct Encoding* of F as

$$\text{Enc}_{Direct}(F) = \left(\sum_{(i,j)\in F} \mathbf{1}_{\mathfrak{P}_i}|_{\text{Up}}, \sum_{(i,j)\in F} \mathbf{1}_{\mathfrak{P}_j}|_{\text{Up}} \right) \in \mathbb{N}^{\text{Up}} \times \mathbb{N}^{\text{Up}}$$

Similarly, we define the *Interfered Encoding* of F as a subset of $\mathbb{Z}^{\text{Up}} \times \mathbb{Z}^{\text{Up}}$ given by

$$\text{Enc}_{Inter}(F) = \sum_{(i,j)\in F} (\pm\mathbf{1})_{\mathfrak{P}_i}|_{\text{Up}} \times \sum_{(i,j)\in F} (\pm\mathbf{1})_{\mathfrak{P}_j}|_{\text{Up}} \subset \mathbb{Z}^{\text{Up}} \times \mathbb{Z}^{\text{Up}}$$

Remark 6.1.8. Here, a set of functions $\mathcal{A}^{\mathcal{B}}$, is restricted to $\mathcal{C} \subset \mathcal{B}$ so that $\mathcal{A}^{\mathcal{B}}|_{\mathcal{C}} = \left\{ f|_{\mathcal{C}}, f \in \mathcal{A}^{\mathcal{B}} \right\}$. And the sum of sets is simply the set of sums from elements of both sets so that $A + B = \{a + b \,|a \in A, b \in B\}$.

To accompany this notion of encoding, we add the notion of decoding.

Definition 6.1.9. Let $Y \in \mathbb{N}^{\text{Up}} \times \mathbb{N}^{\text{Up}}$. We define the *Direct Decoding* of Y as $\text{Dec}_{Direct}(Y) = \text{Enc}^{-1}_{Direct}(Y)$ (the preimage).

Similarly, if $Y \in \mathbb{Z}^{\text{Up}} \times \mathbb{Z}^{\text{Up}}$, we define the *Interfered Decoding* of Y $\text{Dec}_{Inter}(Y)$ as the set of $F \subset \{1, ..., m\}^2$ such that $Y \in \text{Enc}_{Inter}(F)$.

The Encoding/Decoding pairs, give us several equivalence relations over $2^{\{1,...,m\}^2}$.

Definition 6.1.10. Let $F, G \subset \{1, ..., m\}^2$. We say that F and G are *directly indistinguishable* if $\text{Enc}_{Direct}(G) = \text{Enc}_{Direct}(F)$. This is obviously an equivalence relation.

We say that F and G are *always indistinguishable through interference* if

$$\text{Enc}_{Inter}(G) = \text{Enc}_{Inter}(F)$$

We say that F and G are *sometimes indistinguishable through interference* if $\text{Enc}_{Inter}(G) \cap \text{Enc}_{Inter}(F) \neq \emptyset$.

Proposition 6.1.11. *We let $F, G \subset \{1, ..., m\}^2$. Then, F and G are directly indistinguishable if and only if they are always indistinguishable through interference.*

Proof. We must prove

$$\mathrm{Enc}_{Inter}(G) = \mathrm{Enc}_{Inter}(F) \implies \mathrm{Enc}_{Direct}(G) = \mathrm{Enc}_{Direct}(F) \tag{8}$$

and conversely

$$\mathrm{Enc}_{Direct}(G) = \mathrm{Enc}_{Direct}(F) \implies \mathrm{Enc}_{Inter}(G) = \mathrm{Enc}_{Inter}(F) \tag{9}$$

After describing the equivalence classes for direct indistinguishability, we will come back and give a simpler argument in Remark 6.1.25.

For Eq. (8), we first see that for any $F \subset \{1, ..., m\}^2$, the point $\mathrm{Enc}_{Direct}(F)$ is uniquely determined among $\mathrm{Enc}_{Inter}(F)$ as the only point in the preimage of the maximum of the function of type $\mathrm{Enc}_{Inter}(F) \to \mathbb{Z}$ given by $(A, B) \mapsto \sum_{I \in \mathrm{Up}} (A_I + B_I)$. Hence, if $\mathrm{Enc}_{Inter}(G) = \mathrm{Enc}_{Inter}(F)$, the same point is determined in both cases, and so $\mathrm{Enc}_{Direct}(G) = \mathrm{Enc}_{Direct}(F)$.

For Eq. (9), we suppose that $\mathrm{Enc}_{Direct}(G) = \mathrm{Enc}_{Direct}(F)$. Then

$$\left(\sum_{(i,j) \in G} \mathbf{1}_{\Phi_i}|_{\mathrm{Up}}, \sum_{(i,j) \in G} \mathbf{1}_{\Phi_j}|_{\mathrm{Up}} \right) = \left(\sum_{(i,j) \in F} \mathbf{1}_{\Phi_i}|_{\mathrm{Up}}, \sum_{(i,j) \in F} \mathbf{1}_{\Phi_j}|_{\mathrm{Up}} \right).$$

This is an equality which holds for $2|\mathrm{Up}|$ indices. When we consider

$$\sum_{(i,j) \in G} (\pm \mathbf{1})_{\Phi_i}|_{\mathrm{Up}} \times \sum_{(i,j) \in G} (\pm \mathbf{1})_{\Phi_j}|_{\mathrm{Up}},$$

we see that for each term in the sum and each of the $2|\mathrm{Up}|$ indices, the choice of whether we take the function with value $+1$ or -1 is done independent of all the other choices. We have for any $I \in \mathrm{Up}$, that $\left[\sum_{(i,j) \in G} (\pm \mathbf{1})_{\Phi_i}|_{\mathrm{Up}}\right]_I$ only depend on $\left[\sum_{(i,j) \in G} \mathbf{1}_{\Phi_i}|_{\mathrm{Up}}\right]_I$. Indeed, $\left[\sum_{(i,j) \in G} \mathbf{1}_{\Phi_i}|_{\mathrm{Up}}\right]_I$ is the number of $(i, j) \in G$ such that $[\mathbf{1}_{\Phi_i}|_{\mathrm{Up}}]_I = 1$. Therefore, when we assign a sign to these crosses at position I, we will

have

$$\left[\sum_{(i,j)\in G} (\pm\mathbf{1})_{\Phi_i}|_{\mathrm{Up}}\right]_I = \left\{ n_+ - n_- \,\middle|\, n_+ + n_- = \left[\sum_{(i,j)\in G} \mathbf{1}_{\Phi_i}|_{\mathrm{Up}}\right]_I \right\}.$$

To summarize, we have that the choices for any single I depends only on

$$\left[\sum_{(i,j)\in G} \mathbf{1}_{\Phi_i}|_{\mathrm{Up}}\right]_I = \left[\sum_{(i,j)\in F} \mathbf{1}_{\Phi_i}|_{\mathrm{Up}}\right]_I$$

and that $\sum_{(i,j)\in G} (\pm\mathbf{1})_{\Phi_i}|_{\mathrm{Up}}$ is simply the combinations of such choices.

It follows that $\sum_{(i,j)\in G} (\pm\mathbf{1})_{\Phi_i}|_{\mathrm{Up}} = \sum_{(i,j)\in F} (\pm\mathbf{1})_{\Phi_i}|_{\mathrm{Up}}$. □

Remark 6.1.12. As the name of the concepts would suggest, *always indistinguishable* implies *sometimes indistinguishable*. The converse isn't true in general, and this is an obvious consequence of Proposition 6.1.26.

Before we keep building the theory, let us connect it with Hadamard matrices.

Lemma 6.1.13. *For any matrix $M \in \mathcal{M}_m(\pm)$ and any $i, j \in \{1, ..., m\}$,*

$$\frac{1}{2}\left[\mathrm{Flip}_{M,i,j}{\mathrm{Flip}_{M,i,j}}^T - MM^T\right]_{\mathrm{Up}} \in (\pm\mathbf{1})_{\Phi_i}|_{\mathrm{Up}}$$

and

$$\frac{1}{2}\left[\mathrm{Flip}^T_{M,i,j}\mathrm{Flip}_{M,i,j} - M^TM\right]_{\mathrm{Up}} \in (\pm\mathbf{1})_{\Phi_j}|_{\mathrm{Up}}$$

where

$$\left[\mathrm{Flip}_{M,i,j}\right]_{kl} = \begin{cases} -[M]_{ij} & i = k \text{ and } j = l \\ [M]_{kl} & \text{Otherwise} \end{cases}$$

Proof. We have that the k-th row of $\mathrm{Flip}_{M,i,j}$ is the same as that of M if $k \neq i$. Therefore, taking the scalar product of two unchanged rows, we get the same result for $\mathrm{Flip}_{M,i,j}$ and M.($\left[\mathrm{Flip}_{M,i,j}{\mathrm{Flip}_{M,i,j}}^T\right]_{kl} = \left[MM^T\right]_{kl}$ for $k \neq i$ and $l \neq i$)

However, if $k = i$ or $l = i$, but $k \neq l$, then only one of the two rows differ between $\mathrm{Flip}_{M,i,j}$ and M. Because the rows are vectors in $(\pm 1)^m$, this means that the scalar products will differ by either ± 2. [In the sum that is the scalar product,

a 1 becomes a -1 or vice-versa; in total, the change is ± 2.] Therefore, we see that $\left[\text{Flip}_{M,i,j}\text{Flip}_{M,i,j}{}^T\right]_{kl} - \left[MM^T\right]_{kl} \in \{2, -2\}$ if $k \neq l$ and either $k = i$ or $l = i$.

Finally, we have that, on the diagonal, $\left[\text{Flip}_{M,i,j}\text{Flip}_{M,i,j}{}^T\right]_{kk} = \left[MM^T\right]_{kk}$, because every $v \in (\pm 1)^m$ satisfies $v \cdot v = m$.

This means that $\frac{1}{2}\left[\text{Flip}_{M,i,j}\text{Flip}_{M,i,j}{}^T - MM^T\right]_{\text{Up}} \in (\pm\mathbf{1})_{\Phi_i}|_{\text{Up}}$. The same arguments, for the columns rather than rows, yields

$$\frac{1}{2}\left[\text{Flip}_{M,i,j}{}^T\text{Flip}_{M,i,j} - M^TM\right]_{\text{Up}} \in (\pm\mathbf{1})_{\Phi_j}|_{\text{Up}}$$

□

Proposition 6.1.14. *If we let $M, M' \in \mathcal{M}_m(\pm)$, we have that*

$$\left(\frac{1}{2}\left[M'M'^T - MM^T\right]_{\text{Up}}, \frac{1}{2}\left[M'^TM' - M^TM\right]_{\text{Up}}\right)$$

is an element of the Interfered Encoding of $\text{Flipset}(M, M')$.

Proof. Let $n = |\text{Flipset}(M, M')|$, and let $(i_k, j_k)_{k=1}^n$ a sequence such that

$$\{(i_k, j_k)\,|k \in \{1, ..., n\}\} = \text{Flipset}(M, M')$$

We therefore can define a sequence of matrices $\left(M^{(k)}\right)_{k=0}^n$ such that

$$\begin{aligned} M^{(0)} &= M \\ \forall_{k\in\{1,...,n\}}.M^{(k)} &= \text{Flip}_{M^{(k-1)},i_k,j_k} \end{aligned}$$

Of course, $M^{(n)} = M'$. Then,

$$\begin{aligned} M'M'^T - MM^T &= M^{(n)}M^{(n)T} - M^{(0)}M^{(0)T} \\ &= \sum_{k=1}^{n}\left(M^{(k)}M^{(k)T} - M^{(k-1)}M^{(k-1)T}\right) \\ &= \sum_{k=1}^{n}\left(\text{Flip}_{M^{(k-1)},i_k,j_k}\text{Flip}_{M^{(k-1)},i_k,j_k}{}^T - M^{(k-1)}M^{(k-1)T}\right) \end{aligned}$$

Similarly for $M'^T M' - M^T M$.

The result follows from Lemma 6.1.13. □

Corollary 6.1.15. *If we let $M \in \mathcal{M}_m(\pm)$, and $H \in \mathcal{H}_m$, then*

$$\left(\frac{1}{2}\left[MM^T\right]_{\mathrm{Up}}, \frac{1}{2}\left[M^T M\right]_{\mathrm{Up}}\right)$$

belongs to the Interfered Encoding of $\mathrm{Flipset}(M, H)$.

In particular, for every $F \in \mathrm{Flipset}(M)$,

$$\left(\frac{1}{2}\left[MM^T\right]_{\mathrm{Up}}, \frac{1}{2}\left[M^T M\right]_{\mathrm{Up}}\right) \in \mathrm{Enc}_{Inter}(F)$$

Proposition 6.1.16. *If we let $m > 2$. If $F, G \subset \{1, ..., m\}^2$ with $|F| = |G| = 1$(they are singletons), such that F and G are sometimes indistinguishable through interference, then $G = F$.*

Furthermore, there exists an algorithm which, given $(I, J) \in \mathrm{Enc}_{Inter}(F)$, produces F with $\mathcal{O}(m)$ operations (which implies that it doesn't look at every one of the $\Theta(m^2)$ entries in (I, J)).

In other words, under the condition that $|F| = |G| = 1$, the Interfered Encoding process is invertible (it is loss-less), by a fast procedure (it is efficient). Indeed, we give this procedure explicitly below.

To show this, we will use a few lemmas.

Lemma 6.1.17. *Let m be the dimension, i, j such that $1 \leq i < j \leq m$, two integers, and $l \in \{1, ..., m\}$ arbitrary. Then the following equations hold:*

$$\maltese_i \cap \maltese_j \cap \mathrm{Up} = \{(i, j)\} \tag{10}$$

$$\maltese_i = \dot{\cup}_{l \neq i} \maltese_i \cap \maltese_l \tag{11}$$

$$(i, j) \in \maltese_l \cap \mathrm{Up} \iff (l = i \text{ or } l = j) \tag{12}$$

Proof. By definition, we see that $\{(i, j)\} \subseteq \maltese_i$ and $\{(i, j)\} \subseteq \maltese_j$, so

$$\{(i, j)\} \subseteq \maltese_i \cap \maltese_j$$

If a pair is within $\maltese_i \cap \maltese_j$, then one of its components must be i and the other one must be j. This means $\maltese_i \cap \maltese_j = \{(i,j),(j,i)\}$. Since we intersect with Up, and $i < j$, we have Eq. (10).

Equation (12) follows directly from the definition of a cross.

For Eq. (11), let $k \notin \{i,j\}$. We start by considering the intersection

$$(\maltese_i \cap \maltese_j) \cap (\maltese_i \cap \maltese_k)$$

By Eq. (10) and symmetry, this is

$$\{(i,j),(j,i)\} \cap \{(i,k),(k,i)\} = \emptyset$$

Therefore, the union $\cup_{l \neq i} \maltese_i \cap \maltese_l$ is disjoint, which will allow us to do a counting argument. It is also obvious that

$$\maltese_i \supseteq \cup_{l \neq i} \maltese_i \cap \maltese_l \tag{13}$$

We therefore have two finite sets one within the other. The number of elements in $\maltese_i$ is $|\maltese_i| = 2(m-1)$. The number of elements in $\cup_{l \neq i} \maltese_i \cap \maltese_l$ is

$$\begin{aligned} |\cup_{l \neq i} \maltese_i \cap \maltese_l| &= \sum_{l \neq i} |\maltese_i \cap \maltese_l| \\ &= \sum_{l \neq i} 2 \\ &= 2(m-1) \end{aligned}$$

By the inclusion (13), we thus establish Eq. (11). □

We also want to define an algorithm to perform the decoding.

Algorithm D (*Detects single cross*). Invocation:

$$j \leftarrow \text{Detect}(i, \text{Cross}, \text{Enc})$$

Let $m > 2$ the dimension, $i, k \in \{1, ..., m\}$ two indices,

$$\text{Cross} \in \left(\{1, ..., m\}^2\right)^{m-1}$$

and $\text{Enc} \in \{-1, 0, 1\}^{\text{Up}}$. If $\text{Cross}_{[in]}$ is an ordered list of the elements of $\maltese_i \cap \text{Up}$, and $\text{Enc}_{[in]} \in (\pm\mathbf{1})_{\maltese_k}|_{\text{Up}}$, we produce $j_{[out]} = k$.

Furthermore, we do so with work $\Upsilon \in \mathcal{O}(m)$.

D1. [First element.] Set $k \leftarrow 1$, and $(a, b) \leftarrow \text{Cross}_k$.

D2. [Zero?] If $\text{Enc}_{ab} = 0$, set $s \leftarrow 0$. (We have seen 0 non-zero elements so far.)

D3. [Non-Zero?] If $\text{Enc}_{ab} \neq 0$, set $s \leftarrow 1$, and $j \leftarrow \begin{cases} a & a \neq i \\ b & \text{Otherwise} \end{cases}$. (We have seen 1 non-zero element so far; the correct value of j is either its current value or it is i.)

D4. [Second element.] Set $k \leftarrow 2$, and $(a, b) \leftarrow \text{Cross}_k$.

D5. [Zero?] If $\text{Enc}_{ab} = 0$ and $s = 1$, terminate. (If there is a non-zero element somewhere, then j already has the correct value, since it was either j or i, and now we know it isn't i.)

D6. [Non-Zero twice?] If $\text{Enc}_{ab} \neq 0$ and $s = 1$, then set $s \leftarrow 2$, $j \leftarrow i$ and terminate. (We have seen 2 non-zero elements so far. They are all non-zero, and so i is the correct value.)

D7. [Non-Zero once?] If $\text{Enc}_{ab} \neq 0$ and $s = 0$, then set $j \leftarrow \begin{cases} a & a \neq i \\ b & \text{Otherwise} \end{cases}$ and terminate. (There are some zero elements, so this non-zero element is the only one.)

D8. [Next element.] Set $k \leftarrow k + 1$, and $(a, b) \leftarrow \text{Cross}_k$

D9. [Non-Zero?] If $\text{Enc}_{ab} \neq 0$, set $j \leftarrow \begin{cases} a & a \neq i \\ b & \text{Otherwise} \end{cases}$ and terminate. (There are some zero elements, so this non-zero element is the only one.)

D10. [Zero?] If $\text{Enc}_{ab} = 0$, go back to D8. ∎

Proof. (of Proposition 6.1.16.) Let $m > 2$. By contradiction, let us assume that we have $G \neq F$ with $|G| = |F| = 1$ such that there is $(I, J) \in \mathrm{Enc}_{Inter}(F) \cap \mathrm{Enc}_{Inter}(G)$.

Since $F = \{(i_1, j_1)\}$ and $G = \{(i_2, j_2)\}$, we must have $(i_1, j_1) \neq (i_2, j_2)$ and hence either $i_1 \neq i_2$ or $j_1 \neq j_2$. Without loss of generality, suppose $i_1 \neq i_2$.

Then, by hypothesis

$$I \in (\pm\mathbf{1})_{\maltese_{i_1}}|_{\mathrm{Up}} \cap (\pm\mathbf{1})_{\maltese_{i_2}}|_{\mathrm{Up}}$$

By definition, this means I is supported on $\maltese_i \cap \maltese_j \cap \mathrm{Up}$ and by Eq. (10), this means that I contains exactly 1 non-zero element. But because $I \in (\pm\mathbf{1})_{\maltese_{i_1}}|_{\mathrm{Up}}$, we know that it contains $m - 1$ non-zero elements. Therefore, since $m > 2$, we have a contradiction.

Furthermore, we can call $j \leftarrow \mathrm{Detect}(i, \mathrm{Cross}, \mathrm{Enc})$ once with $i_{[in]} = 1$, $\mathrm{Cross}_{[in]} = \maltese_1 \cap \mathrm{Up}$, and $\mathrm{Enc}_{[in]} = I$ to get $j_{[out]} = a$ and once with $i_{[in]} = 1$, $\mathrm{Cross}_{[in]} = \maltese_1 \cap \mathrm{Up}$, and $\mathrm{Enc}_{[in]} = J$ to get $j_{[out]} = b$. Then, we will have that $F = \{(a, b)\}$. And the total of operations will be $2\mathcal{O}(m) = \mathcal{O}(m)$. □

Remark 6.1.18. We have $f^2 = \mathbf{1}_{\mathcal{A}}$ for all $f \in (\pm\mathbf{1})_{\mathcal{A}}$, and so by taking the absolute value of the Encoding, we see that $\mathrm{Enc}_{Inter}(F)$ carries the same information as $\mathrm{Enc}_{Direct}(F)$ for $|F| = 1$. What will happen is that as $|F|$ grows, $\mathrm{Enc}_{Direct}(F)$ will still be more or less decodable whereas $\mathrm{Enc}_{Inter}(F)$ will be less so.

Definition 6.1.19. If we let $F \in 2^{\{1,\ldots,m\}^2}$, for convenience, we define

$$\mathrm{Tally}_{row}(F), \mathrm{Tally}_{col}(F) \in \mathbb{N}^m$$

as

$$\begin{aligned} [\mathrm{Tally}_{row}(F)]_i &= \sum_{(i,l)\in F} 1 \\ [\mathrm{Tally}_{col}(F)]_j &= \sum_{(k,j)\in F} 1 \end{aligned}$$

or in other words,

$$[\mathrm{Tally}_{row}(F)]_i = |\{(k,l) \in F \,|k = i\}|$$

and

$$[\mathrm{Tally}_{col}(F)]_j = |\{(k,l) \in F \,|l = j\}|$$

And let $\mathcal{R}, \mathcal{C} \in \mathbb{N}^m$ such that $\sum \mathcal{R}_i = \sum \mathcal{C}_i$, we define

$$\mathrm{Flipset}_{\mathcal{R},\mathcal{C}} = \left\{G \subset \{1,...,m\}^2 \,|\mathcal{R} = \mathrm{Tally}_{row}(G) \text{ and } \mathcal{C} = \mathrm{Tally}_{col}(G)\right\}$$

The following shows how easy things would be without interference.

Proposition 6.1.20. *Let $m > 2$. If $F \subset \{1,...,m\}^2$, then the set of $G \subset \{1,...,m\}^2$ directly indistinguishable from F is precisely* $\mathrm{Flipset}_{\mathrm{Tally}_{row}(F),\mathrm{Tally}_{col}(F)}$. *In other words, the equivalence classes correspond directly with pairs $\mathcal{R}, \mathcal{C} \in \mathbb{N}^m$ such that $\sum_i \mathcal{R}_i = \sum_j \mathcal{C}_j$.*

Furthermore, given (I, J), the Direct Encoding of F, there exists an algorithm which returns $\mathrm{Tally}_{row}(F), \mathrm{Tally}_{col}(F)$ *in $\mathcal{O}(m)$ operations.*

Once more, we require a few lemmas.

Lemma 6.1.21. *Let $m > 2$, and $\mathcal{T} \in \mathbb{N}^m$, we have, for all $(j,k) \in \mathrm{Up}$,*

$$\left[\sum_i \mathcal{T}_i \mathbf{1}_{\maltese_i}|_{\mathrm{Up}}\right]_{jk} = \mathcal{T}_j + \mathcal{T}_k$$

Proof. It is immediate from Eq. (12) that $(j,k) \in \maltese_i$ implies that either $j = i$ or $k = i$. Hence, if $[\mathbf{1}_{\maltese_i}|_{\mathrm{Up}}]_{jk} \neq 0$, then either $j = i$ or $k = i$, and so, in the sum $[\sum_i \mathcal{T}_i \mathbf{1}_{\maltese_i}|_{\mathrm{Up}}]_{jk} = \sum_i \mathcal{T}_i [\mathbf{1}_{\maltese_i}|_{\mathrm{Up}}]_{jk}$, only $\mathcal{T}_j + \mathcal{T}_k$ remains. □

We also want an algorithm.

Algorithm T (*Tally an Encoding*). Invocation:

$$\mathrm{Tally} \leftarrow \mathrm{TallyDecode}(\mathrm{Enc})$$

Let $m > 2$, $\text{Enc} \in \mathbb{N}^{\text{Up}}$, and $\mathcal{T} \in \mathbb{N}^m$. Given $\text{Enc}_{[in]} = \sum_i \sum_{j=1}^{\mathcal{T}_i} \mathbf{1}_{\maltese_i}|_{\text{Up}} = \sum_i \mathcal{T}_i \mathbf{1}_{\maltese_i}|_{\text{Up}}$, we produce $\text{Tally}_{[out]} = \mathcal{T}$.

Furthermore, we do so with work $\Upsilon \in \mathcal{O}(m)$.

T1. [First Tally.] Set $k \leftarrow 1$, and $\text{Tally}_1 \leftarrow \frac{1}{2}\left([\text{Enc}]_{12} + [\text{Enc}]_{13} - [\text{Enc}]_{23}\right)$. (After this, $\text{Tally}_1 = \mathcal{T}_1$, since $\mathcal{T}_1 = \frac{1}{2}\left((\mathcal{T}_1 + \mathcal{T}_2) + (\mathcal{T}_1 + \mathcal{T}_3) - (\mathcal{T}_2 + \mathcal{T}_3)\right)$, which is in turn

$$\frac{1}{2}\left(\left[\sum_i \mathcal{T}_i \mathbf{1}_{\maltese_i}|_{\text{Up}}\right]_{12} + \left[\sum_i \mathcal{T}_i \mathbf{1}_{\maltese_i}|_{\text{Up}}\right]_{13} - \left[\sum_i \mathcal{T}_i \mathbf{1}_{\maltese_i}|_{\text{Up}}\right]_{23}\right)$$

following Lemma 6.1.21 and thus $\frac{1}{2}\left([\text{Enc}]_{[in]12} + [\text{Enc}]_{[in]13} - [\text{Enc}]_{[in]23}\right)$.)

T2. [Next element.] Set $k \leftarrow k + 1$. If $k > m$, terminate.

T3. [Current Tally.] Set $\text{Tally}_k \leftarrow [\text{Enc}]_{1,k} - \text{Tally}_1$, and go back to T2. (After this, $\text{Tally}_k = \mathcal{T}_k$, since $\mathcal{T}_k = (\mathcal{T}_1 + \mathcal{T}_k) - \mathcal{T}_1$ and thus $[\text{Enc}]_{[in]1,k} - \mathcal{T}_1$.) ∎

Remark 6.1.22. Just like Algorithm D, Algorithm T could have been given in terms of any cross $\maltese_i \cap \text{Up}$, but to simplify the presentation, we have used $\maltese_1 \cap \text{Up}$ directly.

Lemma 6.1.23. *Let $m > 2$, $\mathcal{T} \in \mathbb{N}^m$. If $I = \sum_i \sum_{j=1}^{\mathcal{T}_i} \mathbf{1}_{\maltese_i}|_{\text{Up}}$, then there are no other $\mathcal{S} \neq \mathcal{T} \in \mathbb{N}^m$ such that $I = \sum_i \sum_{j=1}^{\mathcal{S}_i} \mathbf{1}_{\maltese_i}|_{\text{Up}}$.*

Proof. If this were violated, then Algorithm T would have to return two different outputs, which is impossible. □

Lemma 6.1.24. *Let $m > 2$, $G \in \{1, ..., m\}^2$. Then, the Direct Encoding of G will be*

$$\left(\sum_i \left[\text{Tally}_{row}(G)\right]_i \mathbf{1}_{\maltese_i}|_{\text{Up}}, \sum_j \left[\text{Tally}_{col}(G)\right]_j \mathbf{1}_{\maltese_j}|_{\text{Up}}\right)$$

Proof. This is direct from the definitions. For instance,

$$\begin{aligned}\sum_{(i,j)\in G} \mathbf{1}_{\maltese_i}|_{\text{Up}} &= \sum_i \sum_j \mathbf{1}_G((i,j))\, \mathbf{1}_{\maltese_i}|_{\text{Up}} \\ &= \sum_i \left[\text{Tally}_{row}(G)\right]_i \mathbf{1}_{\maltese_i}|_{\text{Up}}\end{aligned}$$

□

Proof. (of Proposition 6.1.20.) From the previous two lemmas, we have that the tallies determine and are determined by the Direct Encoding. So for $F, G \subset \{1, ..., m\}^2$, having the same tallies and being directly indistinguishable is the same thing.

Finally, we can use Algorithm T twice as we did in the proof of proposition 6.1.16 to get the result. □

Remark 6.1.25. Armed with this characterization of direct indistinguishably, we see that Eq. (9) has a very simple proof. Indeed, it suffices to show that if $F, G \in \text{Flipset}_{\mathcal{R},\mathcal{C}}$ for some tallies $\mathcal{R}, \mathcal{C} \in \mathbb{N}^m$ such that $\sum_i \mathcal{R}_i = \sum_j \mathcal{C}_j$, then $\text{Enc}_{Inter}(G) = \text{Enc}_{Inter}(F)$.

We see that

$$\text{Enc}_{Inter}(G) = \sum_i \sum_{j=1}^{\mathcal{R}_i} (\pm\mathbf{1})_{\maltese_i}|_{\text{Up}} \times \sum_j \sum_{i=1}^{\mathcal{C}_j} (\pm\mathbf{1})_{\maltese_j}|_{\text{Up}}$$

and by the same argument,

$$\text{Enc}_{Inter}(F) = \sum_i \sum_{j=1}^{\mathcal{R}_i} (\pm\mathbf{1})_{\maltese_i}|_{\text{Up}} \times \sum_j \sum_{i=1}^{\mathcal{C}_j} (\pm\mathbf{1})_{\maltese_j}|_{\text{Up}}$$

So the Encoding/Decoding process is "well-behaved" when there is no "interference." In particular, if there were no interference in the context of Corollary 6.1.15, then we could compute $|\text{Flipset}(M)| = d(M, \mathcal{H}_m)$ using Algorithm T, but since there is interference, we need something more sophisticated.

Proposition 6.1.26. *Let $\mathcal{R}, \mathcal{C} \in \mathbb{N}^m$ such that $\sum_i \mathcal{R}_i = \sum_j \mathcal{C}_j$ and let*

$$Y \in \sum_i \sum_{j=1}^{\mathcal{R}_i} (\pm\mathbf{1})_{\maltese_i}|_{\text{Up}} \times \sum_j \sum_{i=1}^{\mathcal{C}_j} (\pm\mathbf{1})_{\maltese_j}|_{\text{Up}}$$

Then, $\text{Dec}_{Inter}(Y)$ contains $\text{Flipset}_{\mathcal{R},\mathcal{C}}$.

Furthermore, letting $\mathcal{I}, \mathcal{J} \subset \{1, ..., m\}$ with $|\mathcal{I}| = |\mathcal{J}| = 2$, if $F \in \text{Dec}_{Inter}(Y)$ with F disjoint from $(\mathcal{I} \times \mathcal{J})$, then $F \cup (\mathcal{I} \times \mathcal{J}) \in \text{Dec}_{Inter}(Y)$.

Proof. That $\text{Dec}_{Inter}(Y)$ contains $\text{Flipset}_{\mathcal{R},\mathcal{C}}$ is simply a consequence of Eq. (9).

For the rest, let $O : \mathrm{Up} \to \{0\}$ be the constant function identically 0. Then, we have that $(O, O) \in \mathrm{Enc}_{Inter}(\mathcal{I} \times \mathcal{J})$ for

$$\mathcal{I} \times \mathcal{J} = \{(i_1, j_1), (i_1, j_2), (i_2, j_1), (i_2, j_2)\}$$

as above. Indeed, we can choose $O = \mathbf{1}_{\maltese_{i_1}} - \mathbf{1}_{\maltese_{i_1}} + \mathbf{1}_{\maltese_{i_2}} - \mathbf{1}_{\maltese_{i_2}} \in \sum_{(i,j)\in\mathcal{I}\times\mathcal{J}} (\pm\mathbf{1})_{\maltese_i}$ for instance.

Also, if $A \cap B = \emptyset$, $\mathrm{Enc}_{Inter}(A) + \mathrm{Enc}_{Inter}(B) = \mathrm{Enc}_{Inter}(A \cup B)$. From which we get $Y = (O, O) + Y \in \mathrm{Enc}_{Inter}(\mathcal{I} \times \mathcal{J}) + \mathrm{Enc}_{Inter}(F) = \mathrm{Enc}_{Inter}(F \cup (\mathcal{I} \times \mathcal{J}))$ and therefore $F \cup (\mathcal{I} \times \mathcal{J}) \in \mathrm{Dec}_{Inter}(Y)$. □

This tells us, roughly speaking, that there will be no useful *upper* bound on n such that $\mathrm{Dec}_{Inter}(Y)$ contains $\mathrm{Flipset}_{\mathcal{R},\mathcal{C}}$ with $\sum_i \mathcal{R}_i = \sum_j \mathcal{C}_j = n$. There will, however, be some useful *lower* bounds.

6.2 Computing Lower Bounds on $d(M, \mathcal{H}_m)$

For a given $Y \in \mathbb{Z}^{\mathrm{Up}} \times \mathbb{Z}^{\mathrm{Up}}$, we want to find n_{low} such that if $F \in \mathrm{Dec}_{Inter}(Y)$, $n_{low} \leq |F|$. If M is a matrix in $\mathcal{M}_m(\pm)$ and $Y = \left(\frac{1}{2}\left[MM^T\right]_{\mathrm{Up}}, \frac{1}{2}\left[M^TM\right]_{\mathrm{Up}}\right)$, then, since $\mathrm{Flipset}(M) \subset \mathrm{Dec}_{Inter}(Y)$, we would have $n_{low} \leq d(M, \mathcal{H}_m)$. This will be key to our study of the (ε, δ)-Quasi-Hadamard matrices.

We begin with a partial substitute for Lemma 6.1.21.

Proposition 6.2.1. *Let $\mathcal{R}, \mathcal{C} \in \mathbb{N}^m$ such that $\sum_i \mathcal{R}_i = \sum_j \mathcal{C}_j$ and let*

$$(I, J) \in \sum_i \sum_{j=1}^{\mathcal{R}_i} (\pm\mathbf{1})_{\maltese_i}|_{\mathrm{Up}} \times \sum_j \sum_{i=1}^{\mathcal{C}_j} (\pm\mathbf{1})_{\maltese_j}|_{\mathrm{Up}}$$

We have, for every (i, j) such that $i < j$:

$$\begin{aligned} \left|[I]_{ij}\right| &\leq \mathcal{R}_i + \mathcal{R}_j \\ \left|[I]_{ij}\right| &\equiv \mathcal{R}_i + \mathcal{R}_j \mod 2 \end{aligned}$$

similarly for the columns:

$$\begin{aligned} \left|[J]_{ij}\right| &\leq \mathcal{C}_i + \mathcal{C}_j \\ \left|[J]_{ij}\right| &\equiv \mathcal{C}_i + \mathcal{C}_j \mod 2 \end{aligned}$$

Proof. Similarly to what we had for Lemma 6.1.21, $[I]_{ij}$ is a sum of $\mathcal{R}_i + \mathcal{R}_j$ non-zero terms, each of which could be either $+1$ or -1. Thus, $[I]_{ij} = n_+ - n_-$ with $n_+ + n_- = \mathcal{R}_i + \mathcal{R}_j$ for two non-negative integers n_+ and n_-. The fact that $\left|[I]_{ij}\right| \leq n_+ + n_- = \mathcal{R}_i + \mathcal{R}_j$ is immediate. The parity comes from

$$n_+ + n_- = n_+ - n_- + 2n_- \equiv n_+ - n_- \mod 2$$

and the absolute value preserves parity.

The same argument show the result for J as well. □

In fact, we can use this parity constraint to construct an algorithm to partition the is and js according to the parity of $\mathcal{R}_i$ and $\mathcal{R}_j$, etc..

Algorithm P (*Partition* $\{1, ..., m\}$ *w.r.t. Encoding Parity*). Invocation:

$$a, A, b, B \leftarrow \text{ParityClass}\,(\text{Enc})$$

with $a, b \in \{0, ..., m\}, A, B \in \{1, ..., m\}^m$ and $\text{Enc} \in \mathbb{Z}^{\text{Up}}$. We let $\mathcal{R} \in \mathbb{N}^m$. Given $\text{Enc} \in \sum_i \sum_{j=1}^{\mathcal{R}_i} (\pm\mathbf{1})_{\Phi_i}|_{\text{Up}}$, we will produce $\mathcal{A} = \{A_i \,|i \leq a\}$ and $\mathcal{B} = \{B_i \,|i \leq b\}$ such that $\mathcal{A} \cup \mathcal{B} = \{1, ..., m\}$, $\mathcal{A} \cap \mathcal{B} = \emptyset$, and such that either $\mathcal{A} = \{i \,|\mathcal{R}_i \text{ is even.}\}$ or $\mathcal{B} = \{i \,|\mathcal{R}_i \text{ is even.}\}$. In other words, such that $\{\mathcal{A}, \mathcal{B}\} = \{\{i \,|\mathcal{R}_i \text{ is even.}\}, \{i \,|\mathcal{R}_i \text{ is odd.}\}\}$.

P1. [Initialize.] Set $a \leftarrow 1, b \leftarrow 0$, $A_a \leftarrow 1$, and $k \leftarrow 2$. (We put $1 \in \mathcal{A}$.)

P2. [Congruent?] If Enc_{1k} is even, then set $a \leftarrow a + 1$, and $A_a \leftarrow k$. (After this step, $k \in \mathcal{A}$ if and only if $\mathcal{R}_1 \equiv \mathcal{R}_k \mod 2$.)

P3. [Not congruent?] If Enc_{1k} is odd, then set $b \leftarrow b + 1$, and $B_b \leftarrow k$. (After this step, $k \in \mathcal{B}$ if and only if $\mathcal{R}_1 \not\equiv \mathcal{R}_k \mod 2$.)

P4. [Increment.] Set $k \leftarrow k + 1$. If $k > m$, terminate.

P5. [Loop.] If $k \leq m$, go back to P2. ∎

Now, we develop an algorithm which automatically extracts information from

$$\left(\frac{1}{2}\left[MM^T\right]_{\mathrm{Up}}, \frac{1}{2}\left[M^TM\right]_{\mathrm{Up}}\right)$$

and produces a lower bound for us.

Algorithm L (*Finds a lower bound for $|G|$ in the Interfered Decoding of a given Encoding.*). We recommend that the reader look at the proof immediately below this algorithm before going through the steps here. The proof explains the logic behind the algorithm. Once we know what the algorithm is trying to do, and why this yields a lower bound, it suffices to check that the steps do indeed follow the logic of the proof. Furthermore, the precise steps add nothing, from the mathematical point of view. If the logic behind the algorithm outlined in the proof is convincing to the reader, then the steps given here can be regarded as an implementation detail, and safely skipped. In particular, we shall not discuss the performance of the algorithm, aside from the remark that the number of conjunctions will "blow up" quite fast unless MaxTotal is chosen small enough.

The statement which is proven is quite simple: If we let $\mathcal{T} \in \mathbb{Z}^m$ and Parity $\in \{0,1\}^m$ provided that $\forall_i.\mathcal{T}_i \equiv \mathrm{Parity}_i \mod 2$, and $\forall_{i<j}.0 \leq [\mathrm{Enc}]_{ij} \leq \mathcal{T}_i + \mathcal{T}_j$ and $[\mathrm{Enc}]_{ij} \equiv \mathcal{T}_i + \mathcal{T}_j \mod 2$, then we will have, after the algorithm terminates, that LowerBound $\leq \sum_i \mathcal{T}_i$.

Let $\mathsf{Odd} = 1, \mathsf{Even} = 0, \mathsf{GEQ} = 0, \mathsf{EQ} = 1, \mathsf{True} = 1, \mathsf{False} = 0$.

L01. [Initialize.] $[\![\forall_j]\!]$, set $\mathrm{OldDis}_{0j} \leftarrow \big(\mathsf{GEQ}, \mathrm{Parity}_j\big)$.

L02. Set OldCount $\leftarrow 1$.

L03. [The outer loop.] Set $i \leftarrow 1$.

L04. [Loop over cases.] Set Value $\leftarrow \mathrm{Parity}_i$, and Count $\leftarrow 0$. (We consider all nontrivial possibilities for $\mathcal{T}_i$. We start with $\mathcal{T}_i = \mathrm{Parity}_i$, and then $\mathcal{T}_i = \mathrm{Parity}_i + 2$, etc... until the conjunction we get is trivial. We also count the conjunctions using the variable Count.)

L05. [Assume nontrivial.] Set NonTrivial $\leftarrow$ False. (We start a given case assuming the conjunction will be trivial, and we must establish during the process that it isn't.)

L06. [New case.] $[\![\forall_j]\!]$, do the following: Set

$$\text{NewDis}_{\text{Count},j} \leftarrow \begin{cases} (\mathsf{EQ}, \text{Value}) & i = j \\ \big(\mathsf{GEQ}, \text{Enc}_{\min(i,j)\max(i,j)} - \text{Value}\big) & \text{Otherwise} \end{cases}$$

If both $j \neq i$ and $[\text{NewDis}_{\text{Count},j}]_2 \geq 2$, then set NonTrivial $\leftarrow$ True. (By hypothesis, $\mathcal{T}_j \geq \text{Enc}_{\min(i,j)\max(i,j)} - \mathcal{T}_i$ for $j \neq i$, and we set $\mathcal{T}_i = \text{Value}$. This is only one of the possibilities of course, but the disjunction of all the nontrivial conjunctions together with the first trivial conjunction is valid. If we get a condition $\mathcal{T}_j \geq c$ with $c < 2$, it is trivial. If all the conditions are trivial, the whole conjunction is trivial. Because we loop over ascending values of Value, we will have seen all nontrivial conjunctions before the first trivial conjunction.)

L07. [Trivial?] If NonTrivial = False, then set $\text{NewDis}_{\text{Count},i} \leftarrow (\mathsf{GEQ}, \text{Value})$. (For the first trivial Conjunction, instead of $\mathcal{T}_i = \text{Value}$, set $\mathcal{T}_i \geq \text{Value}$, therefore covering all remaining possibilities.)

L08. [Loop (inner).] If NonTrivial = True, then set Value $\leftarrow$ Value + 2, Count $\leftarrow$ Count + 1, and go back to L05.

L09. [Reset Count.] Set NewCount $\leftarrow$ Count, then Count $\leftarrow 0$. (At this point, the New disjunction is true, and contains Count conjunctions.)

L10. [Loop.] Set Old $\leftarrow 0$. (This loop is over all conjunctions in Old and New. This is how we take the conjunction of two disjunctions of conjunctions of Equalities/Inequalities for $\mathcal{T}$. Every conjunction from the Old is combined with every conjunction from the New. Since these conjunctions have the same structure, we know how to compute the result. Then, we end up with a huge disjuction of conjunctions (MergeDis), which we simplify by removing duplicates, and by removing invalid conjunctions (conjunctions which are always false). We also remove conjunctions which imply a lower bound bigger than MaxTotal, as we need some way of limiting the time the algorithm takes to terminate.)

L11. [Loop.] Set New $\leftarrow 0$.

L12. [Mini loop.] Set Valid $\leftarrow \mathsf{True}$, and $k \leftarrow 1$. (We compute the conjunctions of two sets of conjunctions by merging the conditions component by component, i.e. those for $\mathcal{T}_1$ together, and then those for $\mathcal{T}_2$, etc..)

L13. [Alias.] Set $(S_1, V_1) \leftarrow \mathrm{OldDis}_{\mathrm{Old},k}$, and $(S_2, V_2) \leftarrow \mathrm{NewDis}_{\mathrm{New},k}$. (The S_x means either $\geq$ or $=$ and the V_x is just a number. We have $\mathrm{OldConditions}\,(k)_{\mathcal{T}_k} = (\mathcal{T}_k\, S_1\, V_1)$ and $\mathrm{NewConditions}\,(k)_{\mathcal{T}_k} = (\mathcal{T}_k\, S_2\, V_2)$. What follows is simply a case by case merging into a single condition.)

L14. [$\mathcal{T}_k = V_1 \wedge \mathcal{T}_k = V_1$.] If $S_1 = S_2 = \mathsf{EQ}$ and $V_1 = V_2$, then set $\mathrm{MergeDis}_{\mathrm{Count},k} \leftarrow (\mathsf{EQ}, V_1)$. (This is redundant and we can just say $\mathcal{T}_k = V_1$.)

L15. [$\mathcal{T}_k = V_1 \wedge \mathcal{T}_k = V_2$ with $V_1 \neq V_2$.] If $S_1 = S_2 = \mathsf{EQ}$ and $V_1 \neq V_2$, then set Valid $\leftarrow \mathsf{False}$. (This is a contradiction.)

L16. [$\mathcal{T}_k \geq V_1 \wedge \mathcal{T}_k \geq V_2$.] If $S_1 = S_2 = \mathsf{GEQ}$, then set

$$\mathrm{MergeDis}_{\mathrm{Count},k} \leftarrow (\mathsf{GEQ}, \max(V_1, V_2))$$

(Only the biggest bound matters.)

L17. [$\mathcal{T}_k = V_1 \wedge \mathcal{T}_k \geq V_2$.] If $S_1 = \mathsf{EQ}$ and $S_2 = \mathsf{GEQ}$, then set $\mathrm{MergeDis}_{\mathrm{Count},k} \leftarrow (\mathsf{EQ}, \max(V_1, V_2))$. If $V_1 < V_2$, then set Valid $\leftarrow \mathsf{False}$. (Obviously, this is a contradiction if $V_1 < V_2$. Otherwise, the equality is superfluous.)

L18. [$\mathcal{T}_k \geq V_1 \wedge \mathcal{T}_k = V_2$.] If $S_1 = \mathsf{GEQ}$ and $S_2 = \mathsf{EQ}$, then set $\mathrm{MergeDis}_{\mathrm{Count},k} \leftarrow (\mathsf{EQ}, \max(V_1, V_2))$. If $V_2 < V_1$, then set Valid $\leftarrow \mathsf{False}$.

L19. [Contradiction?] $k \leftarrow k + 1$. If Valid $= \mathsf{True}$ and $k \leq m$, then go back to L13. (We stop if we reach a contradiction. Otherwise we merge completely.)

L20. [Remove duplicates.] If Valid $= \mathsf{True}$ and $\exists_{\mathrm{PreCount}<\mathrm{Count}} \forall_k . \mathrm{MergeDis}_{\mathrm{Count},k} = \mathrm{MergeDis}_{\mathrm{PreCount},k}$, then set Valid $\leftarrow \mathsf{False}$. (We do not consider duplicates valid. They are, but we do not want to add them to the new disjuction. Setting Valid $\leftarrow \mathsf{False}$ will ensure that we do not add this one.)

L21. [Cull Bounds.] If Valid $= \mathsf{True}$ and $\sum_k \left[\mathrm{MergeDis}_{\mathrm{Count},k}\right]_2 > \mathrm{MaxTotal}$, then set Valid $\leftarrow \mathsf{False}$. (We reject any conjunction which implies a lower bound on

$\sum_k \mathcal{T}_k$ greater than MaxTotal. Indeed, implicitly in every disjuction is the case where $\sum_k \mathcal{T}_k >$ MaxTotal. This allows us to limit the growth of the number of conjunctions.)

L22. [Commit Conjunction.] If Valid = True, then set Count $\leftarrow$ Count + 1. (Only "commit" the current conjunction to the disjuction if it was valid. Otherwise, do not change Count, which will result in the overwriting of the current space with a valid conjunction or in the exclusion of the current space from the set of conjunctions included in MergeDis.)

L23. [Loop.] Set New $\leftarrow$ New + 1. If New $\leq$ NewCount $-$ 1, then go back to L12. (Loop over $\forall_{\text{Old}\in\{0,\ldots,\text{OldCount}-1\}}\forall_{\text{New}\in\{0,\ldots,\text{NewCount}-1\}}$.)

L24. [Same] Set Old $\leftarrow$ Old + 1. If Old $\leq$ OldCount $-$ 1, then go back to L11.

L25. [Copy.] $[\![\forall_{\text{Merge}<\text{Count}}\forall_k]\!]$, set $\text{OldDis}_{\text{Count},k} \leftarrow \text{MergeDis}_{\text{Count},k}$. Set OldCount $\leftarrow$ Count. (We accumulate the Merged disjuction into the Old, so that the final result will be the conjunction of all disjunctions seen as New for every i-th cross in Enc.)

L26. [Loop] Set $i \leftarrow i + 1$. If $i \leq m$, then go back to L04.

L27. [Get Lower Bound.] Set

$$\text{LowerBound} \leftarrow \min\left(\text{MaxTotal}, \min_{\{0,\ldots,\text{OldCount}-1\}} \sum_k \left[\text{OldDis}_{\text{Old},k}\right]_2\right)$$

(The lower bound on $\sum_k \mathcal{T}_k$ is simply the lowest bound among all conjunctions. If it is equal to MaxTotal, one might want to re-execute the algorithm with a higher value for MaxTotal, as one might get a higher lower bound.) ∎

Proof. As before, we take $\mathcal{T} \in \mathbb{Z}^m, \text{Parity} \in \{0,1\}^m$. We begin with the logical propositions that $\forall_i.\mathcal{T}_i \equiv \text{Parity}_i \mod 2$, and $\forall_{i<j}.0 \leq [\text{Enc}]_{ij} \leq \mathcal{T}_i + \mathcal{T}_j \wedge [\text{Enc}]_{ij} \equiv \mathcal{T}_i + \mathcal{T}_j \mod 2$, together with $\sum_i \mathcal{T}_i \geq$ MaxTotal. We take

$$\begin{aligned}\text{Dis0} &= \left((\forall_i.\mathcal{T}_i \equiv \text{Parity}_i \mod 2) \wedge \left(\forall_{i<j}.\,[\text{Enc}]_{ij} \leq \mathcal{T}_i + \mathcal{T}_j\right)\right.\\ &\wedge \left.\left(\forall_{i<j}.0 \leq [\text{Enc}]_{ij}\right) \wedge \left(\forall_{i<j}.\,[\text{Enc}]_{ij} \equiv \mathcal{T}_i + \mathcal{T}_j \mod 2\right)\right)\end{aligned}$$

and the assumption of the algorithm is that Dis0 is true, therefore,

$$\text{Dis0} \vee \left(\sum_i \mathcal{T}_i \geq \text{MaxTotal} \right)$$

is also true.

Set $\text{Dis} = \wedge_i \left(\mathcal{T}_i \geq \text{Parity}_i \right)$ (a disjunction of conjunctions with only one conjunction). We see that $\text{Dis0} \vee (\sum_i \mathcal{T}_i \geq \text{MaxTotal})$ is equivalent to

$$(\text{Dis0} \wedge \text{Dis}) \vee \left(\sum_i \mathcal{T}_i \geq \text{MaxTotal} \right)$$

Then, we proceed to walk through every cross. Fixing i, we look at

$$[\text{Enc}]_{\min(i,j)\,\max(i,j)} \leq \mathcal{T}_i + \mathcal{T}_j$$

We set

$$\begin{aligned} \text{Dis2} \quad &= \quad \left(\vee_{x \in V} \left(\left(\mathcal{T}_1 \geq [\text{Enc}]_{\min(i,1)\,\max(i,1)} - x \right) \wedge (\cdots) \wedge (\mathcal{T}_i = x) \wedge (\cdots) \right) \right) \\ &\vee \quad \left(\left(\mathcal{T}_1 \geq [\text{Enc}]_{\min(i,1)\,\max(i,1)} - x_{max} \right) \wedge (\cdots) \wedge (\mathcal{T}_i \geq x_{max}) \wedge (\cdots) \right) \end{aligned}$$

where V is a set of values with parity Parity_i such that $\max V + 2 = x_{max}$ and x_{max} is the smallest value with parity Parity_i such that the conjunction of

$$\left(\left(\mathcal{T}_1 \geq [\text{Enc}]_{\min(i,1)\,\max(i,1)} - x_{max} \right) \wedge (\cdots) \wedge (\mathcal{T}_i = x_{max}) \wedge (\cdots) \right)$$

with $\wedge_j \left(\mathcal{T}_i \geq \text{Parity}_i \right)$ is equivalent to $\wedge_j \left(\mathcal{T}_i \geq \text{Parity}_i \right)$. (It is the first valid value which makes the corresponding conjunction trivial given $\wedge_j \left(\mathcal{T}_i \geq \text{Parity}_i \right)$.)

Then we get $\text{Dis3} = \text{Dis2} \wedge \text{Dis}$, which we rewrite as a disjunction. We see that $\text{Dis0} \vee (\sum_i \mathcal{T}_i \geq \text{MaxTotal})$ is equivalent to $(\text{Dis0} \wedge \text{Dis3}) \vee (\sum_i \mathcal{T}_i \geq \text{MaxTotal})$. Furthermore, let Dis4 be Dis3 with all terms which imply $(\sum_i \mathcal{T}_i \geq \text{MaxTotal})$ dropped.

Then, we still have Dis0 $\vee$ $(\sum_i \mathcal{T}_i \geq \text{MaxTotal})$ equivalent to

$$(\text{Dis0} \wedge \text{Dis4}) \vee \left(\sum_i \mathcal{T}_i \geq \text{MaxTotal} \right)$$

Therefore, we set Dis $\leftarrow$ Dis4, and repeat for the next value of i.

We see that at the end, it is still true that (Dis0 $\wedge$ Dis) $\vee$ $(\sum_i \mathcal{T}_i \geq \text{MaxTotal})$ is equivalent to Dis0 $\vee$ $(\sum_i \mathcal{T}_i \geq \text{MaxTotal})$. If (Dis0 $\wedge$ Dis) is false, then

$$\left(\sum_i \mathcal{T}_i \geq \text{MaxTotal} \right)$$

must be true. Furthermore, if there exists n_{min} such that every term in the disjunction implies $\sum_i \mathcal{T}_i \geq n_{min}$, then $\sum_i \mathcal{T}_i \geq n_{min}$ is consistent with the initial assumptions, namely that $\forall_i . \mathcal{T}_i \equiv \text{Parity}_i \mod 2$, and $\forall_{i<j} . 0 \leq [\text{Enc}]_{ij} \leq \mathcal{T}_i + \mathcal{T}_j \wedge [\text{Enc}]_{ij} \equiv \mathcal{T}_i + \mathcal{T}_j \mod 2$.

This is the idea behind Algorithm L. With it, it should be clear that the specific steps indeed follow this logic. □

Remark 6.2.2. By using Algorithm P, we can get two vectors $\mathcal{P}(1), \mathcal{P}(2) \in \{0,1\}^m$ such that one of them is $\forall_k . [\mathcal{P}(i)]_k \equiv \mathcal{T}_k \mod 2$. Therefore, by calling Algorithm L with these two choices for $\text{Parity}_{[in]}$, we will get two values $\mathcal{L}_1 = \text{LowerBound}_{[out]}$ (for $\text{Parity}_{[in]} = \mathcal{P}(1)$) and $\mathcal{L}_2 = \text{LowerBound}_{[out]}$ (for $\text{Parity}_{[in]} = \mathcal{P}(2)$) such that $\min(\mathcal{L}_1, \mathcal{L}_2) \leq \sum_k \mathcal{T}_k$.

Furthermore, if we have $(I, J) \in \mathbb{Z}^{\text{Up}} \times \mathbb{Z}^{\text{Up}}$, we can call Algorithm L four times (twice with $\text{Enc}_{[in]} = |I|$ and twice with $\text{Enc}_{[in]} = |J|$). We get two lower bounds of the form $\min(\mathcal{L}_1, \mathcal{L}_2)$ as above, which we call $\mathcal{L}_{row}, \mathcal{L}_{col}$. We thus have that if G is in the Interfered Decoding of (I, J), then $|G| \geq \max(\mathcal{L}_{row}, \mathcal{L}_{col})$.

By doing this for $(I, J) = \left(\frac{1}{2} \left[MM^T \right]_{\text{Up}}, \frac{1}{2} \left[M^T M \right]_{\text{Up}} \right)$, we get $d(M, \mathcal{H}_m) \geq \max(\mathcal{L}_{row}, \mathcal{L}_{col})$, a lower bound on the distance between a given matrix and the Hadamard matrices.

6.3 Properties and Examples of (ε, δ)-Quasi-Hadamard matrices

With the algorithms and propositions we have presented, it is possible to make quantitative claims about (ε, δ)-Quasi-Mins of various functions over $\mathcal{M}_m(\pm)$. However, we still need a bit more considerations before we can *judge* these quantitative claims. We recall that, for instance for the function E_{rows}, we say (if $\mathcal{H}_m \neq \emptyset$) $q \in \mathcal{M}_m(\pm)$ is a (ε, δ)-Quasi-Min if

$$|E_{rows}(q)| < \varepsilon$$

and yet

$$d_{hamming}(q, \mathcal{H}_m) \geq \delta$$

where E_{rows}^{-1} is simply the preimage. (ε, δ)-Quasi-Mins of E_{rows} over $\mathcal{M}_m(\pm)$ shall be called (ε, δ)-Quasi-Hadamard matrices. In order to judge if a (ε, δ)-Quasi-Min is trivial or not, we must introduce a scale to which we might compare ε and δ.

Proposition 6.3.1. *For $M \in \mathcal{M}_m(\pm)$, the following hold*

1. $\max E_{rows}(\mathcal{M}_m(\pm)) = \frac{m(m-1)}{2} m^2$. *(It is reached on $M_{ij} = +1, \forall_{i,j}$.)*
2. *If $d(M, \mathcal{H}_m) = 1$, then $E_{rows}(M) = 4(m-1)$.*
3. $\max_{M \in \mathcal{M}_m(\pm)} d_{hamming}(M, \mathcal{H}_m) = \frac{m(m-1)}{2}$. *(It is reached on $M_{ij} = +1, \forall_{i,j}$.)*
4. *For $D \in \mathbb{N}$, $|\mathcal{B}_D(M)| = \sum_{i=0}^{D-1} \binom{m^2}{i}$.*

Remark 6.3.2. As we see, even matrices $M \in \mathcal{M}_m(\pm)$ with $d(M, \mathcal{H}_m) = 1$ have $E_{rows}(M) \in \Theta_4(m)$. This means that, in the limit $m = 2^n, n \to \infty$, even matrices at a trivial distance from the Hadamard matrices will have a divergent energy. (Here, we take $m = 2^n$ simply because we know that $\mathcal{H}_{2^n} \neq \emptyset$ for every n.)

We also see that the maximal value of the energy is $\Theta(m^4)$, which grows very fast.

Furthermore, we see that even at a moderate distance from $\mathcal{H}_m$, the number of matrices closer than $\mathcal{H}_m$ will be enormous.

For all of these reasons, we can get a more meaningful way of comparing Quasi-Hadamard matrices for various values of m by introducing the following.

Definition 6.3.3. We define the *Quasi-Hadamard Number* $\mathrm{QH} : \mathcal{M}_m(\pm) \setminus \mathcal{H}_m \to \mathbb{R}$ by

$$\mathrm{QH}(M) = \frac{E_{Quasi}(M)}{d(M, \mathcal{H}_m)}$$

where E_{Quasi} was defined in Definition 4.4.1.

Remark 6.3.4. Note that if $F \in \mathrm{Flipset}(M)$, then

$$\begin{aligned} \mathrm{QH}(M) &= \frac{\frac{1}{(m-1)} \sum_{x \in \mathrm{Up}} \left| \left[\frac{1}{2} M M^T \right]_x \right|}{\frac{1}{(m-1)} \sum_{x \in \mathrm{Up}} \left[\sum_{(i,j) \in F} \mathbf{1}_{\mathbb{X}_i}|_{\mathrm{Up}} \right]_x} \\ &= \frac{\sum_{x \in \mathrm{Up}} \left| \left[\frac{1}{2} M M^T \right]_x \right|}{\sum_{x \in \mathrm{Up}} \left[\sum_{(i,j) \in F} \mathbf{1}_{\mathbb{X}_i}|_{\mathrm{Up}} \right]_x} \end{aligned}$$

and, of course $\frac{1}{2} M M^T \in \sum_{(i,j) \in F} (\pm \mathbf{1})_{\mathbb{X}_i}|_{\mathrm{Up}}$. This connects QH to our previous discussion of Interfered and Direct Encoding. Indeed, we see that for $\mathrm{QH}(M) \approx 0$, we have an element of the Interfered Encoding which is much less in norm l_1 than the Direct Encoding. In other words, a small $\mathrm{QH}(M)$ implies a lot of "interference."

Furthermore, we see that the biggest value of $\mathrm{QH}(M)$ if $\mathcal{H}_m \neq \emptyset$ is 1.

Finally, notice that $\mathrm{QH}(M)$ cannot be computed exactly in general, but we get the exact numerator and a lower bound on the denominator, which gives an upper bound on $\mathrm{QH}(M)$.

Proposition 6.3.5. *For $M \in \mathcal{M}_m(\pm)$, the following hold*

1. $\max E_{Quasi}(\mathcal{M}_m(\pm)) = \frac{m^2}{4}$ *, and it is reached on the matrix M with $+1$s everywhere. ($M_{ij} = +1, \forall_{i,j}$)*

2. *If $d(M, \mathcal{H}_m) = 1$, then $E_{Quasi}(M) = 1$.*

And now, we use, as promised, Algorithm L to capture a key property of a matrix obtained using the algorithms discussed in Chapter 5. **This is an example of what the tools we have developed allow us to prove.**

Theorem 6.3.6. *In dimension $m = 20$, for $X = \mathcal{M}_m(\pm)$, $f_1 = E_{rows}, f_2 = E_{det}, f_3 = E_{Quasi}, f_4 = E_{col}$, there exists a point $x \in X$ with $\mathrm{QH}(x) \leq \frac{3}{19} < 0.16$ such that*

$$\begin{aligned} E_{rows}(x) &= 288 \\ E_{Quasi}(x) &= \frac{36}{19} \approx 1.89 \\ E_{col}(x) &= 288 \\ E_{det}(x) &= -69337088 \times 10^5 \\ d(x, \mathcal{H}_m) &\geq 12 \end{aligned}$$

In other words, it is an $(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, D)$-Co-Quasi-Min for the functions $f_1, ..., f_4$, the hamming distance, and X such that

$$\begin{aligned} \varepsilon_1 &= 288 \\ \varepsilon_2 &= 33062912 \times 10^5 \approx 0.32288 \left|\min E_{det}(X)\right| \\ \varepsilon_3 &= \frac{36}{19} \approx 1.89 \\ \varepsilon_4 &= 288 \\ D &\geq 12 \end{aligned}$$

Remark 6.3.7. Note that ε_2, though seemingly big, is only 33 percent of the absolute value of $|E_{det}(H)| = 20^{\frac{20}{2}}$ for $H \in \mathcal{H}_{20}$. If the absolute value of the determinant is viewed as the hyper-volume of a cube, then we have that the relative difference between one side of the cube corresponding to x and one side of the cube corresponding to $H \in \mathcal{H}_{20}$ is $1 - \left(\frac{6933708800000}{20^{\frac{20}{2}}}\right)^{\frac{1}{20}} \approx 0.019$, or about 2 percent.

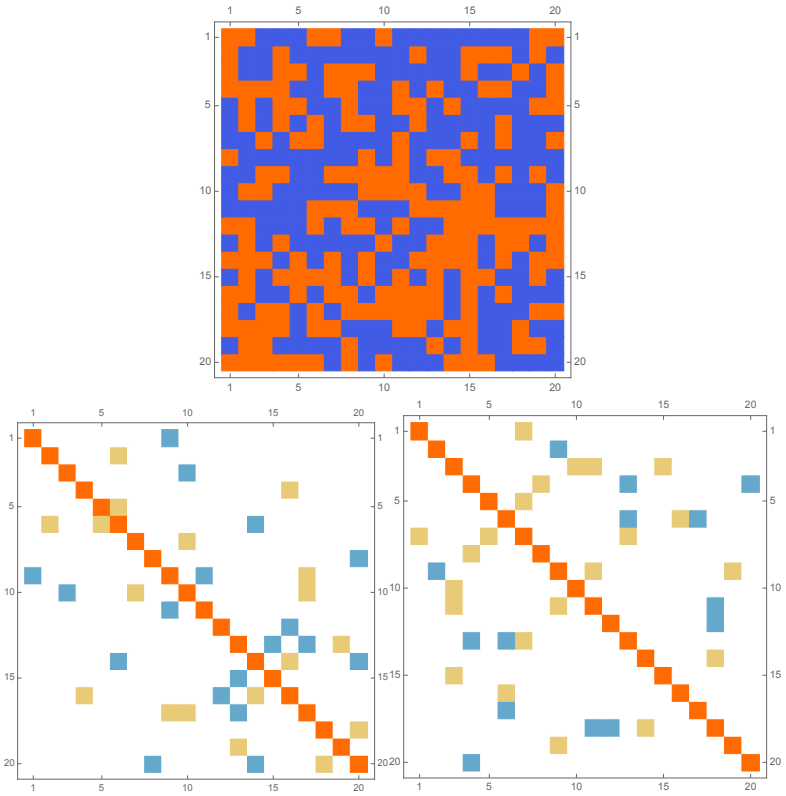
Also notice that $\varepsilon_3 < 2E_{Quasi}(M)$ for $d(M, \mathcal{H}_m) = 1$. In other words, notice how, despite being at a distance ≥ 12 from the nearest Hadamard, meaning that $|\{y \in \mathcal{M}_m(\pm) \,|\, d(y, x) < d(x, \mathcal{H}_m)\}| > 10^{20}$, $E_{Quasi}(x)$ is comparable to the energy of matrices adjacent to $\mathcal{H}_m$.

Furthermore, note that since $E_{rows}(x) = E_{cols}(x)$, we have that x^T, which is not equal to x, is another $(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, D)$-Co-Quasi-Min for the same values of $\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, D$.

Proof. We give such a matrix in figure 6.7. Verifying the values it takes for $f_1, ..., f_4$ is just a matter of computation, and the lower bound on $d(x, \mathcal{H}_m)$ is given by executing Algorithm L. □

Remark 6.3.8. Of course, finding such a matrix was the hard part. This matrix was obtained from the optimization algorithms developed in previous chapters. More specifically, we have used both Multi-Metro semantics, have systematically tried various temperatures (thermal energies), and have used a switching approach between $E_{rows}, E_{det}, E_{col}$, etc.. The same formula was able to produce Hadamard matrices for $m = 20$ approximately 0.5 percent of the time for short optimization runs (approximately 1000 to 10000 steps), but since all such matrices are already known for $m < 36$, showing such a matrix wouldn't attest to the claim or add to the knowledge about Hadamard matrices (we have therefore not shown such matrices).

With the algorithms developed in this thesis, longer runs could be tried, in higher dimensions, to find yet more impressive Co-Quasi-Mins or even new Hadamard matrices. Some additional insights or a revolution in computing power would be needed to find a real Hadamard matrix for a dimension such as $m = 668$ where their existence is still in question, but the case of complex Hadamard matrices may yield more easily, since much still isn't known even for single digits dimensions. (see [9].)



Above is the element of $\mathcal{M}_m(\pm)$ itself. Below, to the left is the symmetric product MM^T. To the right, M^TM. White means 0. Above, red means $+1$, blue means -1. Below, all the non-zero elements not on the diagonal have an absolute value of 4.

Figure 6.7: A Co-Quasi-Min.

Chapter 7

Conclusion

Now that we have discussed basic optimization, given some old methods, as well as some new ones, parallelized such methods with a form of reduction, that we have discussed Hadamard matrices (real and complex), and characterized them with optimization problems, that we have developed fast algorithms to implement the best methods we could find, and used these algorithms to find interesting elements of $\mathcal{M}_m(\pm)$, and finally that we have investigated the presence of Quasi-Mins for the function E_{rows}, developed tools to prove some of their properties, and used such tools to establish now non-faithful all of these functions which characterize the Hadamard matrices are, we will discuss a few ideas which might form the basis of future research. We have already mentioned some of these in the context of previous chapters, but what follows is tangentially related to what came before, and so we list these ideas in no particular order.

- Looking at [8], we see that instead of maintaining $\Phi(M)$ as we change individual components, it is possible to maintain the LU Decomposition directly. It isn't clear how to incorporate this within our algorithms, but it might be possible to do so.

- There are connections between the Gradient Descent and Hadamard matrices which are interesting yet almost totally unrelated to what we have seen in our work, see [32]. Indeed, Hadamard matrices can be used to *solve gradient descent*

problems with neural networks.

- Investigations of the encoding Φ used in Chapter 5 reveal that it is possible to compute the necessary arithmetic in a bit-wise fashion, and this is indeed what we have done for the case $q = 2$ (real Hadamard). Such ideas have been discovered before, see [18], and much more can be done, see [29, pp. 147-202], for instance. This quickly becomes impractical when $q \to \infty$, but the cases $q = 2, 3, 4, 5, 6, 7, 8$ could probably all be explored with some practical benefits.

- Some different approaches have been tried, to find Hadamard matrices using optimization, see [15]. The techniques are almost entirely different from those developed here. Therefore, it could be interesting to try and combine these techniques. Furthermore, we note the lack of benchmarks in the literature to compare new methods for finding Hadamard matrices. We have provided the asymptotic performance of our algorithms, but a future direction might be to implement all our methods, all methods in [15], together with a catalog of other methods we have tried but discarded, and get a comparison between the actual time each method takes to find a Hadamard matrix on some specified hardware.

- It should be possible to use the equivalence relation between matrices $\mathcal{M}_m(\pm)$ induced by the action of $G_m \times G_m$ to reduce the size of the search space. A simple trick is to fix the first row to be uniformly $+1$, the second row to be $+1$ for the first half, and -1 for the second half, and the third row to be alternating $\frac{m}{4}$ $+1$s followed by $\frac{m}{4}$ -1s , repeated twice. We could still optimize, while imposing this constraint strictly. However, this does not fully use the equivalence relation, since even with this constraint, there will be many distinct possibilities within the same equivalence class. It would be interesting to remove some of that redundancy. On that note, there are many more operations which leave $\mathcal{H}_m$ invariant, for instance see [37]. It would be interesting to extend such operations to $\mathcal{M}_m(\pm)$ such that they leave $\frac{1}{2}(E_{rows} + E_{col})$ and E_{det} constant, and then incorporate those into our algorithms.

- So far, we have used a fixed selection procedure. Using Reinforcement Learning,

it is possible to learn such a procedure, when given some stimuli (the neighboring energies), and some actions to perform (flipping an element). See for instance [45]. The question then would be what kind of selection method would be learned in the context of finding Hadamard matrices.

- Our lower bounds on the minimum distance from $\mathcal{H}_m$ simply relies on the properties of Interfered Indicator functions of crosses being added together. One step further would be to replace Interfered Indicators by random variables $\mathcal{X}_{\mathcal{A}} \in (\pm\mathbf{1})_{\mathcal{A}}^{\Omega}$ distributed as

$$\sum_{f \in (\pm\mathbf{1})_{\mathcal{A}}} \frac{1}{\left|(\pm\mathbf{1})_{\mathcal{A}}\right|} \delta_f = \frac{1}{2^{|\mathcal{A}|}} \sum_{f \in (\pm\mathbf{1})_{\mathcal{A}}} \delta_f$$

With this, we could consider the Uniform Encoding given by

$$\mathrm{Enc}_{Unif}(F) = \left(\sum_{(i,j) \in F} (\!| \mathcal{X}_{\boxplus_i} |\!), \sum_{(i,j) \in F} (\!| \mathcal{X}_{\boxplus_j} |\!) \right)$$

(every term $(\!|X|\!)$ is distributed according to X, but is independent from all other such terms.) and then, decoding would be a distribution over $2^{\{1,\ldots,m\}^2}$: $\mathrm{Dec}_{Unif}(Y)$ would be the conditional distribution of $\mathcal{F} \in \left(2^{\{1,\ldots,m\}^2}\right)^{\Omega}$ given that $\mathrm{Enc}_{Unif}(\mathcal{F}) = Y$. With this, we could make sense of claims such as "the probability that F Encoded Y is negligible."

- Both the uniformly random + independent idea and what is studied in the thesis fail to capture the fact that some elements of the Interfered Encodings can never arise as

$$\left(\frac{1}{2}\left[M'M'^T - MM^T\right]_{\mathrm{Up}}, \frac{1}{2}\left[M'^T M' - M^T M\right]_{\mathrm{Up}} \right)$$

for $M, M' \in \mathcal{M}_m(\pm)$. It would be even more restrictive to only consider Interfered Encodings arising from the above with $M' \in \mathcal{H}_m$ such that $d(M, M') =$

$d(M, \mathcal{H}_m)$. As a further investigation, one could generate extremely large numbers of such matrices, observe the distributions of the actual Encodings, and after having approximated the relevant distributions, one could study the properties of encoding and decoding. An even further step would be to start with the Hadamard matrices already known, and gather data about the distribution of

$$\left(\frac{1}{2}\left[M'M'^T - MM^T\right]_{\mathrm{Up}}, \frac{1}{2}\left[M'^T M' - M^T M\right]_{\mathrm{Up}}\right)$$

and Flipset (M, M') for $M' \in \mathcal{H}_m$ such that $d(M, M') = d(M, \mathcal{H}_m)$. This was in fact the original idea; to attack this problem as a data mining problem, but we saw the need for some *provable guarantees*, even if they might be less impressive. Now that we have a reliable, but perhaps sub-optimal alternative, we can build upon it with the probabilistic approach.

- The problem of finding a tight lower bound $n_{min} \leq \sum_i \mathcal{T}_i$ given

$$I \in \sum_i \sum_{j=1}^{\mathcal{T}_i} (\pm \mathbf{1})_{\mathbb{X}_i}$$

is related to the so-called Integer-Linear-Programming Problem. The general form is to find x to maximize $c^T x$ subject to $Ax \leq b$ and $x \geq 0$ and $x \in \mathbb{Z}^n$. How is it related? Well, clearly, if we could maximize $-\sum_i x_i$ (i.e. minimize $\sum_i x_i$) such that $\forall_i x_i \geq 0$, and such that $\forall_{i<j}. -x_i - x_j \leq -\left|[I]_{ij}\right|$ and $\forall_i x_i \equiv \mathcal{T}_i \mod 2$, for $x \in \mathbb{Z}^m$, then, $n_{min} = \sum_i x_i \leq \sum_i \mathcal{T}_i$. This is almost an Integer-Linear-Programming problem, except for $\forall_i x_i \equiv \mathcal{T}_i \mod 2$. For more on the ILP problem, see [50].

- In the context of searching for arbitrary complex Hadamard matrices, it is possible to combine the gradient methods with the kinds of general Multi-Metropolis methods to get something new and exciting. The problem with Gradient Descent is that it is greedy, or locally optimal. Because of this it will get stuck once it reaches a stationary point of the Gradient (for instance a local minimum). However, the Gradient Methods have the advantage that they converge

rapidly to the local minima. This partially compensates for the fact that the underlying space is continuous. In order to combine the non-greedy nature of Multi-Metro with the speed of convergence of Gradient Methods, we can imagine generating candidates, applying a Gradient Descent step to each of them, and selecting from among the results. Because of the convergence speed of the Gradient Descent, we should generate candidates at greater distances from each other. For instance, if $\mathcal{B}$ is the current encoding of the matrix, we can generate $\mathcal{B} + \sum_i C_{ij} e_{ij}$ for each value of j (m in total), such that $(C_{ij})_{i,j \in \{1,\ldots,m\}}$ are independent, identically distributed random variables with values in the interval $(-\pi, \pi)$. For concreteness, we could imagine a uniform distribution on that interval. Or, we could imagine a distribution with mean 0 and which falls off as a Gaussian on either side of 0. The possibilities for this kind of scheme are numerous and need to be explored before we know what details matter and what details don't. However, such a scheme is also applicable to a wide number of problems where Gradient Methods are available, and perhaps different variations of this idea work best in different applications.

Appendix A

Heuristics

For completeness, we will list some heuristics, though we won't use them in the rest of the thesis.

Optimizers can share some memory either with their descendants or with each other as they optimize. When local minima are remembered to avoid them, we call it "tabu search." When local minima are remembered to focus the efforts in promising regions of the landscape, we call it "ant-colony," or "beehive," or "particle swarm" search. For more on this, see [41].

Optimizers can share resources during the generation of candidates or during the selection. They can also use multiple current states to generate the candidates. Such methods fall under the umbrella of "genetic algorithms," "memetic algorithms," or "evolutionary methods". For some further readings, see [44, 41, 10].

If they have parameters, Optimizers can influence each other's parameters. The simplest case for Metropolis-Hastings is that a single optimizer iterates as its thermal energy diminishes until the algorithm gets stuck. This is called "simulated annealing."[41] When the change in thermal energy is predetermined independently of the states visited, we call it a "cooling schedule." It is possible to measure the performance of the optimizer and adjust the thermal energy in consequence, however.

At a larger scale, a whole grid of Optimizers can "try" different parameters, test their performance, and periodically restart with new parameters based on the results. Soon, this iteration become a kind of optimization in itself, where the set of states

is the set of parameters and the energy is the performance of the Optimizers as a function of their parameters. This process of optimizing the parameters of an optimizer is called "meta-optimization." This is one of the heuristics we *will* use to optimize the choice of T, the thermal energy.

Since our main inspiration for navigating and energy landscape has come from thermodynamics, we should mention that the corresponding *quantum theory* (see [36, 14]) contains more possibilities than the classical one. This is not surprising since classical phenomena are simply a subset of quantum mechanics.

In our context, the main difference is how systems get out of local minima.

Classically, they do so by random fluctuations in their configuration. The level of available fluctuations is encapsulated in the thermal energy E_T, and the probability of escaping from a local minimum only depends on the *height* or the depth of the *basin of attraction*. In other words, in order to get out of the valley, we have to climb the mountain. If the mountain is extremely high, getting out might be almost impossible, no matter how *thin* it is.

In quantum mechanics however, the probability of occupying a state does not only depend on the energy *of that state*, but also on the *surrounding energy levels*. If we have a very high and very thin peak surrounded by two very deep valleys, the probability of occupation at the peak is much bigger than what you would get simply by looking at the energy of the peak. You can think of the probability of occupation as propagating through the landscape. It is very high in a deep valley but it still penetrates a little bit into the peak. When it reaches the peak, it decays exponentially with a rate dependent on the height of the peak. This means that if the peak is a really thin, the system might simply *go through*[1] it. For some kinds of optimization problems, this ability is extremely valuable.

Simulations of this quantum process on a classical computer have a hard time, but quantum systems would just do it without any "effort." So why not build a physical system for which the laws of quantum mechanics directly optimize some given[2] function? Such a machine would change the optimization game. As it turns

[1] We say "tunneling."

[2] In the same way that you don't need a new classical computer for every new program, we could build a quantum system reusable for many different optimization problems.

out, people are trying, and there is already a "quantum optimizer" though it is still rudimentary, with less than 2000 qubits in total and only local entanglement.

To understand how we can build a system that uses the laws of quantum mechanics to optimize a function, here is a simple illustration of how we can build a system that uses the laws of classical mechanics to optimize a function.

It is interesting to see this effort evolve, as there are only two ways it can go in the next 20 years: either it fails and people lose interest or it succeeds and changes the field of optimization. For further readings on this idea and the current effort to bring it about, see [36, 14].

Appendix B

Matrix Determinant Lemma and Sherman-Morrison

Lemma B.0.1. *(The Matrix Determinant Lemma, see [38]) If we let* $A \in \mathbb{M}_{m,m}(\mathbb{C})$, $u, v \in \mathbb{M}_{1,m}(\mathbb{C})$ *a square matrix and two column vectors, if* A *is invertible, then the following holds:*

$$\det\left(A + uv^T\right) = \left(1 + v^T A^{-1} u\right) \det(A)$$

Proposition B.0.2. *(The Sherman-Morrison Formula, see [42]) Again, letting* $A \in \mathbb{M}_{m,m}(\mathbb{C})$, $u, v \in \mathbb{M}_{1,m}(\mathbb{C})$, *if* A *is invertible and if* $1 + v^T A^{-1} u \neq 0$, *then the following holds:*

$$\left(A + uv^T\right)^{-1} = A^{-1} - \frac{A^{-1} u v^T A^{-1}}{1 + v^T A^{-1} u}$$

By judiciously choosing u, v we can modify either 1) a single element, 2) a row, or 3) a column of A while computing the updated determinant and the updated inverse according to the two previous formulas.

Proposition B.0.3. *If we let* $A \in \mathbb{M}_{m,m}(\mathbb{C})$, $i, j \in \{1, ..., m\}$, $\alpha \in \mathbb{C}, u, v \in$

$\mathbb{M}_{1,m}(\mathbb{C})$, *if* A *is invertible, the following hold:*

$$\begin{aligned}
\frac{\det(A+\alpha e_{ij})}{\det(A)} &= 1+\alpha\left[A^{-1}\right]_{ji} \\
\frac{\det\left(A+\sum_{i'}u_{i'}e_{i'j}\right)}{\det(A)} &= 1+\sum_{i'}u_{i'}\left[A^{-1}\right]_{ji'} \\
\frac{\det\left(A+\sum_{j'}v_{j'}e_{ij'}\right)}{\det(A)} &= 1+\sum_{j'}v_{j'}\left[A^{-1}\right]_{j'i}
\end{aligned}$$

where $e_{ij}\in\mathbb{M}_{m,m}(\mathbb{C})$ *is such that* $[e_{ij}]_{kl}=\begin{cases}1 & i=k \text{ and } j=l\\ 0 & Otherwise\end{cases}$.

Also, if $1+\alpha\left[A^{-1}\right]_{ji}\neq 0$,

$$\forall_{kl}.\left[(A+\alpha e_{ij})^{-1}\right]_{kl} = \left[A^{-1}\right]_{kl}-\frac{\alpha\left[A^{-1}\right]_{ki}\left[A^{-1}\right]_{jl}}{1+\alpha\left[A^{-1}\right]_{ji}}$$

,

if $1+\sum_{i'}u_{i'}\left[A^{-1}\right]_{ji'}\neq 0$,

$$\forall_{kl}.\left[\left(A+\sum_{i'}u_{i'}e_{i'j}\right)^{-1}\right]_{kl}=\left[A^{-1}\right]_{kl}-\frac{\sum_{i'}u_{i'}\left[A^{-1}\right]_{ki'}\left[A^{-1}\right]_{jl}}{1+\sum_{i'}u_{i'}\left[A^{-1}\right]_{ji'}}$$

if $1+\sum_{j'}v_{j'}\left[A^{-1}\right]_{j'i}\neq 0$,

$$\forall_{kl}.\left[\left(A+\sum_{j'}v_{j'}e_{ij'}\right)^{-1}\right]_{kl}=\left[A^{-1}\right]_{kl}-\frac{\sum_{j'}v_{j'}\left[A^{-1}\right]_{ki}\left[A^{-1}\right]_{j'l}}{1+\sum_{j'}v_{j'}\left[A^{-1}\right]_{j'i}}$$

Proposition B.0.3 allows to recompute the determinant and the inverse of a matrix after we *add* a number to one of its elements. For our purposes however, we need to recompute these things after we *multiply* one of the elements of our matrix by a number. We need to write a multiplication as an addition.

Lemma B.0.4. *If we let* $A,B\in\mathbb{M}_{m,m}(\mathbb{C})$, $i,j\in\{1,...,m\}$ *two matrices and indices*

such that $[A]_{kl} = [B]_{kl}$ *for every* $(k, l) \neq (i, j)$*, then we have*

$$B = A + \left([B]_{ij} - [A]_{ij} \right) e_{ij}$$

This is elementary, but it shows that proposition B.0.3 is enough for any change—additive, multiplicative, whatever—to a single element, a row, or a column.

Appendix C

Gradients, Derivatives, and Stationary Points.

Lemma C.0.1. *Let $\mathcal{B} \in \mathcal{M}_m(\mathbb{R})$, i, j and let* part *the partial function of $E_{rows} \circ \Phi^{-1}$ with respect to the (i,j) component, at $\mathcal{B}$. Let $M = \Phi^{-1}(\mathcal{B})$, then, the derivative $\frac{d}{d\theta}\text{part}|_\theta$ is given by*

$$\begin{aligned}\frac{d}{d\theta}\text{part}|_\theta &= -2\Im\left(e^{\imath\theta}\left[\sum_{k>i}\left([M]_{ij}\,\overline{[M]_{kj}}\sum_{j'}[M]_{ij'}\,\overline{[M]_{kj'}}-1\right)\right.\right.\\ &+ \left.\left.\sum_{k<i}\left([M]_{ij}\,\overline{[M]_{kj}}\sum_{j'}[M]_{kj'}\,\overline{[M]_{ij'}}-1\right)\right]\right)\end{aligned}$$

where $\Im(x)$ is the imaginary part of x.

Proof. This is nothing but a relatively tedious calculation, but here we sketch the steps. First, note that

$$\begin{aligned}E_{rows}(M') &= E_{rows}(M) - \sum_{k\neq i}\left|\sum_{j'}[M]_{\min(i,k),j'}\,\overline{[M]_{\max(k,i)j'}}\right|^2\\ &+ \sum_{k\neq i}\left|\sum_{j'}[M']_{\min(i,k),j'}\,\overline{[M']_{\max(k,i)j'}}\right|^2\end{aligned}$$

where $\Phi(M') = \Phi(M) + \theta e_{ij}$. This is useful since the first two terms, not depending on θ, are constant in part(θ), thus become null after deriving. Also, letting $\forall_{ijk}.\alpha_{ikj} = [M]_{ij}\overline{[M]_{kj}}$, we have that

$$\begin{aligned}\frac{d}{d\theta}\text{part}|_\theta &= \frac{d}{d\theta}\left[\sum_{k\neq i}\left|\sum_{j'}[M']_{\min(i,k),j'}\overline{[M']_{\max(k,i)j'}}\right|^2\right] \\ &= \frac{d}{d\theta}\left[\sum_{i<k}\left|\alpha_{ikj}e^{\imath\theta}+\sum_{j'\neq j}\alpha_{ikj'}\right|^2+\sum_{k<i}\left|\alpha_{kij}e^{-\imath\theta}+\sum_{j'\neq j}\alpha_{kij'}\right|^2\right]\end{aligned}$$

To derive, we simply expand using $|X|^2 = X\overline{X}$. The α_{ikj} are constants with respect to θ, and $\frac{d}{d\theta}e^{a\theta} = ae^{a\theta}, \forall_{a\in\mathbb{C}}$. We rearrange and use the fact that $Y - \overline{Y} = 2\imath\Im(Y)$ to get

$$\begin{aligned}\frac{d}{d\theta}\text{part}|_\theta &= -2\Im\left(e^{i\theta}\left[\sum_{i<k}\sum_{j'\neq j}\overline{\alpha_{ikj'}}\alpha_{ikj}+\sum_{k<i}\sum_{j'\neq j'}\alpha_{kij'}\overline{\alpha_{kij}}\right]\right) \\ &= -2\Im\left(e^{i\theta}\left[\sum_{i<k}\alpha_{ikj}\sum_{j'\neq j}\overline{\alpha_{ikj'}}+\sum_{k<i}\overline{\alpha_{kij}}\sum_{j'\neq j}\alpha_{kij'}\right]\right) \\ &= -2\Im\left(e^{i\theta}\left[\sum_{i<k}\alpha_{ikj}\left(\sum_{j'}\overline{\alpha_{ikj'}}-\overline{\alpha_{ikj}}\right)+\sum_{k<i}\overline{\alpha_{kij}}\left(\sum_{j'}\alpha_{kij'}-\alpha_{kij}\right)\right]\right) \\ &= -2\Im\left(e^{i\theta}\left[\sum_{i<k}\left(\alpha_{ikj}\sum_{j'}\overline{\alpha_{ikj'}}-1\right)+\sum_{k<i}\left(\overline{\alpha_{kij}}\sum_{j'}\alpha_{kij'}-1\right)\right]\right)\end{aligned}$$

We now replace $\forall_{ijk}.\alpha_{ikj} = [M]_{ij}\overline{[M]_{kj}}$ and get the result. □

Lemma C.0.2. *Let $\mathcal{B} \in \mathcal{M}_m(\mathbb{R})$, i, j and let* part *the partial function of $E_{rows} \circ \Phi^{-1}$ with respect to the (i, j) component, at $\mathcal{B}$. Then, the corresponding stationary points*

$\mathcal{B} + \theta e_{ij}$ are given by

$$\begin{aligned}\theta &= \pi n - \text{phase}\left[\sum_{i<k}\left([M]_{ij}\,\overline{[M]_{kj}}\overline{\sum_{j'}[M]_{ij'}\,\overline{[M]_{kj'}}} - 1\right)\right.\\ &+ \left.\sum_{k<i}\left([M]_{ij}\,\overline{[M]_{kj}}\sum_{j'}[M]_{kj'}\,\overline{[M]_{ij'}} - 1\right)\right]\end{aligned}$$

where n is any integer and $\text{phase}\left(Re^{i\phi}\right) = \phi$ *if $R \in \mathbb{R}\setminus\{0\}$ and $\phi \in [0, 2\pi)$.*

Proof. Setting the derivative to 0, we see that the imaginary part of

$$e^{\imath\theta}\left[\sum_{i<k}\left([M]_{ij}\,\overline{[M]_{kj}}\overline{\sum_{j'}[M]_{ij'}\,\overline{[M]_{kj'}}} - 1\right) + \sum_{k<i}\left([M]_{ij}\,\overline{[M]_{kj}}\sum_{j'}[M]_{kj'}\,\overline{[M]_{ij'}} - 1\right)\right]$$

is zero.

Rewriting

$$\sum_{i<k}\left([M]_{ij}\,\overline{[M]_{kj}}\overline{\sum_{j'}[M]_{ij'}\,\overline{[M]_{kj'}}} - 1\right) + \sum_{k<i}\left([M]_{ij}\,\overline{[M]_{kj}}\sum_{j'}[M]_{kj'}\,\overline{[M]_{ij'}} - 1\right)$$

as $Re^{\imath\phi}$, we have $\Im\left(e^{i\theta}e^{\imath\phi}\right) = 0$, which means $\theta + \phi = \pi n$ for some $n \in \mathbb{Z}$. □

Lemma C.0.3. *Let $\mathcal{B} \in \mathcal{M}_m(\mathbb{R})$. Let $M = \Phi^{-1}(\mathcal{B})$, then, the gradient of $E_{rows}\circ\Phi^{-1}$ is given by*

$$\begin{aligned}\forall_{ij}.\left[\nabla\left(E_{rows}\circ\Phi^{-1}\right)\right]_{ij} &= -2\left[\sum_{i<k}\Im\left([M]_{ij}\,\overline{[M]_{kj}}\overline{\sum_{j'}[M]_{ij'}\,\overline{[M]_{kj'}}}\right)\right.\\ &+ \left.\sum_{k<i}\Im\left([M]_{ij}\,\overline{[M]_{kj}}\sum_{j'}[M]_{kj'}\,\overline{[M]_{ij'}}\right)\right]\end{aligned}$$

Proof. This follows directly from lemma C.0.1, simply by setting $\theta = 0$ and noticing that $\Im$ is linear and $\Im(1) = 0$. □

Lemma C.0.4. *Let $\mathcal{B} \in \mathcal{M}_m(\mathbb{R})$, $M = \Phi^{-1}(\mathcal{B})$, i, j and let* part *the partial function of $E_{det^2}\circ\Phi^{-1}$ with respect to the (i,j) component, at B. Then, if M is invertible, the*

derivative $\frac{d}{d\theta}\text{part}|_\theta$ *is given by*

$$\frac{d}{d\theta}\text{part}|_\theta = -\left|\det\left(M\right)\right|^2\left(-2\Im\left(\left[M\right]_{ij}\left[M^{-1}\right]_{ji}e^{\imath\theta}\right)+2\left|\left[M\right]_{ij}\left[M^{-1}\right]_{ji}\right|^2\Im\left(e^{\imath\theta}\right)\right)$$

where $\Im\left(x\right)$ *is the imaginary part of* x*.*

Proof. This is an elementary, if tedious calculation. To reduce the clutter, let $\beta = \left|\det\left(M\right)\right|, \alpha = \left[M\right]_{ij}\left[M^{-1}\right]_{ji}$, then note that

$$\begin{aligned}\text{part}\left(\theta\right) &= E_{det^2}\left(\Phi^{-1}\left(B+\theta e_{ij}\right)\right)\\ &= -\beta^2\left|1+\left(e^{\imath\theta}-1\right)\alpha\right|^2\\ &= -\beta^2\left(1+\left(e^{\imath\theta}-1\right)\alpha\right)\overline{\left(1+\left(e^{\imath\theta}-1\right)\alpha\right)}\end{aligned}$$

We expand and differentiate, and get

$$\frac{d}{d\theta}\text{part}|_\theta = -\beta^2\left(\alpha\imath e^{\imath\theta}-\overline{\alpha}\imath e^{-\imath\theta}+\alpha\overline{\alpha}\left(-\imath e^{\imath\theta}+\imath e^{-\imath\theta}\right)\right)$$

Using the fact that $Y-\overline{Y}=2\imath\Im\left(Y\right)$,

$$\begin{aligned}\frac{d}{d\theta}\text{part}|_\theta &= -\beta^2\imath\left(2\imath\Im\left(\alpha e^{\imath\theta}\right)-2\imath\alpha\overline{\alpha}\Im\left(e^{\imath\theta}\right)\right)\\ &= -\beta^2\left(-2\Im\left(\alpha e^{\imath\theta}\right)+2\alpha\overline{\alpha}\Im\left(e^{\imath\theta}\right)\right)\\ &= -\left|\det\left(M\right)\right|^2\left(-2\Im\left(\left[M\right]_{ij}\left[M^{-1}\right]_{ji}e^{\imath\theta}\right)+2\left|\left[M\right]_{ij}\left[M^{-1}\right]_{ji}\right|^2\Im\left(e^{\imath\theta}\right)\right)\end{aligned}$$

□

Lemma C.0.5. *Let* $\mathcal{B}\in\mathcal{M}_m\left(\mathbb{R}\right)$*,* $M=\Phi^{-1}\left(\mathcal{B}\right)$*,* i,j *and let* part *the partial function of* $E_{det^2}\circ\Phi^{-1}$ *with respect to the* (i,j) *component, at* $\mathcal{B}$*. Then, if* M *is invertible, the corresponding partial stationary points* $\mathcal{B}+\theta e_{ij}$ *are given by*

$$\theta = \pi n - \left[\mathcal{B}\right]_{ij} - \text{phase}\left[\left[M^{-1}\right]_{ji} - \left|\left[M^{-1}\right]_{ji}\right|^2\Phi^{-1}\left(-\left[\mathcal{B}\right]_{ij}\right)\right]$$

where n is any integer and $\text{phase}\left(Re^{i\phi}\right) = \phi$ *if* $R \in \mathbb{R} \setminus \{0\}$ *and* $\phi \in [0, 2\pi)$.

Proof. This uses the same principles as for lemma C.0.2, but starting with lemma C.0.4 rather than lemma C.0.1. □

Lemma C.0.6. *Let* $\mathcal{B} \in \mathcal{M}_m(\mathbb{R})$, $M = \Phi^{-1}(\mathcal{B})$. *Then, if M is invertible, the gradient of* $E_{det^2} \circ \Phi^{-1}$ *is given by*

$$\forall_{ij}. \left[\nabla\left(E_{det^2} \circ \Phi^{-1}\right)\right]_{ij} = 2\left|\det(M)\right|^2 \Im\left(\Phi^{-1}\left(\left[\mathcal{B}\right]_{ij}\right)\left[M^{-1}\right]_{ji}\right)$$

Proof. This follows directly from Lemma C.0.4, simply by setting $\theta = 0$ and noticing that $\Im$ is linear and $\Im(1) = 0$. □

Bibliography

[1] Standford CUDA Lectures. https://www.youtube.com/watch?v=3mjRwd7qVbM&list=PLW-tKlWKtuqYB_gQpigbevohyoSrAwFKZ, 2010. [Accessed: 2016.06.05].

[2] CUDA C Programming Guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz3xjNF51lv, September 2015. [Accessed: 2016.01.19].

[3] M. Acton. CppCon 2014: Mike Acton Data-Oriented Design and C++. https://www.youtube.com/watch?v=rX0ItVEVjHc, 2014. [Accessed: 2016.03.27].

[4] G Ausiello. *Complexity and Approximation Combinatorial Optimization Problems and Their Approximability Properties*. Springer, New York, 1999.

[5] T. Banica, B. Collins, and J-M. Schlenker. On Orthogonal Matrices Maximizing the 1-norm. *Indiana Univ. Math. J.*, 59(3):839–856, 2010. [Accessed: 2016.03.29].

[6] Dimitri Bertsekas. *Convex Optimization Algorithms*. Athena Scientific, Belmont, Mass, 2015.

[7] Robert Bordley and Marco LiCalzi. Decision Analysis Using Targets Instead of Utility Functions. *Decisions in Economics and Finance*, 23(1):53–74, 2000.

[8] Y. Brise. Low Rank Updating of Matrix Factorizations. http://people.inf.ethz.ch/ybrise/data/talks/MSEM20120320.pdf, 2012. [Online and accessed 2016.06.06].

[9] W. Bruzda, W. Tadej, and K. Zyczkowski. Complex Hadamard Matrices. `http://chaos.if.uj.edu.pl/~karol/hadamard/index.php?h=chm_problems`, 2016. [Accessed: 2016.06.06].

[10] Xianshun Chen, Yew-Soon Ong, Meng-Hiot Lim, and Kay Chen Tan. A Multi-Facet Survey on Memetic Computation . *IEEE Transactions on Evolutionary Computation*, 15(5):591–607, 2011.

[11] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms (3rd edition)*. MIT Press and McGraw-Hill, 2009.

[12] David E Culler, Jaswinder Pal Singh, and Anoop Gupta. *Parallel Computer Architecture: a Hardware/Software Approach*. Gulf Professional Publishing, 1999.

[13] P. Danziger. Hamming Distance. `http://www.math.ryerson.ca/~danziger/professor/MTH108/Handouts/codes.pdf`. [Accessed: 2016.01.19].

[14] Vasil S Denchev, Sergio Boixo, Sergei V Isakov, Nan Ding, Ryan Babbush, Vadim Smelyanskiy, John Martinis, and Hartmut Neven. What is the Computational Value of Finite Range Tunneling? *arXiv preprint arXiv:1512.02206*, 2015.

[15] P. Embury and A. Rao. *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes: 17th International Symposium, AAECC-17, Bangalore, India, December 16-20, 2007. Proceedings*, chapter A Path to Hadamard Matrices, pages 281–290. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[16] A. Fog. Software optimization resources. `http://www.agner.org/optimize/`. [Accessed: 2016.01.19].

[17] A. Fog. Instruction tables. `http://www.agner.org/optimize/instruction_tables.pdf`, January 2016. [Accessed: 2016.01.19].

[18] F. Geiringer and D. Shelton. Parallel modulo arithmetic using bitwise logical operations, April 29 2004. US Patent App. 10/296,957.

[19] Mariano. Giaquinta and Enrico Giusti. Quasi-minima. *Annales de l'I.H.P. Analyse non linéaire*, 1(2):79–107, 1984.

[20] W.R. Gilks, S. Richardson, and D.J. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman & Hall/CRC, 1998.

[21] V.K. Gupta, R. Parsad, and A. Dhandapani. Hadamard Matrix (Beta). `http://www.iasri.res.in/design/WebHadamard/webhadamard.htm`. [Accessed: 2016.01.19].

[22] M. Harris. Optimizing Parallel Reduction in CUDA. `https://docs.nvidia.com/cuda/samples/6_Advanced/reduction/doc/reduction.pdf`, 2007. [Accessed: 2016.03.27].

[23] J. Hoffman, C. Johnson, and M. Nazarov. Computational thermodynamics. `http://www.csc.kth.se/~cgjoh/ambsthermo.pdf`, January 2012. [Accessed: 2016.01.19].

[24] Brian W Kernighan, Dennis M Ritchie, and Per Ejeklint. *The C Programming Language*, volume 2. prentice-Hall Englewood Cliffs, 1988.

[25] Hadi Kharaghani and Behruz Tayfeh-Rezaie. Hadamard Matrices of Order 32. *Journal of Combinatorial Designs*, 21(5):212–221, 2013.

[26] D. E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Edition)*. Pearson Education, Inc., Boston, MA, 2013.

[27] D. E. Knuth. *The Art of Computer Programming, Volume 3 (2nd Edition)*. Pearson Education, Inc., Boston, MA, 2013.

[28] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Edition)*. Pearson Education, Inc., Boston, MA, 2014.

[29] D. E. Knuth. *The Art of Computer Programming, Volume 4A (1st Edition)*. Pearson Education, Inc., Boston, MA, 2015.

[30] R. A. Kronmal and A. V. Peterson. On the Alias Method for Generating Random Variables from a Discrete Distribution. *The American Statistician*, 33(4):214–218, 1979.

[31] J. S. Liu, F. Liang, and W. H. Wong. The Multiple-Try Method and Local Optimization in Metropolis Sampling. *Journal of the American Statistical Association*, 95(449):121–134, 2000.

[32] M. Maddren. Hadamard with neural networks. `https://classes.soe.ucsc.edu/cmps290c/Spring13/proj/mmaddren_paper.pdf`, 2013. [Online and accessed 2016.06.06].

[33] Jan Magnus. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. John Wiley, New York, 1999.

[34] T. Masters. *Deep Belief Nets in C++ and CUDA C, Volume I.* CreateSpace, Middletown, DE, 2015.

[35] C. Muratori. Handmade Hero. `https://hero.handmade.network/episodes`, 2014. [Accessed: 2016.06.05].

[36] H. Neven. When can Quantum Annealing Win? `http://googleresearch.blogspot.ca/2015/12/when-can-quantum-annealing-win.html`. [Accessed: 2016.01.27].

[37] William P. Orrick. Switching Operations for Hadamard Matrices. *SIAM Journal on Discrete Mathematics*, 22(1):31–50, 2008.

[38] W. H. Press, B.P. Flannery, S. A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, MA, 1992.

[39] A. Rege. An Introduction to Modern GPU Architecture. `ftp://download.nvidia.com/developer/cuda/seminar/TDCI_Arch.pdf`, 2008. [Accessed: 2016.01.19].

[40] K. Schacke. On the Kronecker Product. `http://www.math.uwaterloo.ca/~hwolkowi/henry/reports/kronthesisschaecke04.pdf`, August 2013. [Accessed: 2016.01.19].

[41] Bart Selman and Carla P Gomes. Hill-climbing Search. *Encyclopedia of Cognitive Science*, 2006.

[42] J. Sherman and W. J. Morrison. Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. *Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[43] N.J.A. Sloane. A Library of Hadamard Matrices. `http://neilsloane.com/hadamard/`. [Accessed: 2016.01.20].

[44] Rainer Storn and Kenneth Price. Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[45] A. Sutton, R. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[46] W. Tadej and K. Zyczkowski. A Concise Guide to Complex Hadamard Matrices. *Open Syst. Inf. Dyn.*, 13:133–177, 2006.

[47] T. Tao. *Topics in Random Matrix Theory*. Americal Mathematical Society, Providence, Rhode Island, 2012.

[48] Lloyd Trefethen and D. Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[49] E. Tressler. A Survey of the Hadamard Conjecture. Master's thesis, Virginia Polytechnic Institute and State University, `http://scholar.lib.vt.edu/theses/available/etd-05042004-120929/unrestricted/thesis_revised.pdf`, 2004. [Accessed: 2016.01.19].

[50] H Paul Williams. *Logic and Integer Programming*, volume 130. Springer, Berlin, 2009.