Methods for Estimating the Diagonal of Matrix Functions

Jesse Harrison Laeuchli

Williamsburg, Virginia

Bachelor of Science, University of Notre Dame, 2007
Master of Science, College of William and Mary, 2012

A Dissertation presented to the Graduate Faculty
of the College of William and Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

The College of William and Mary
May 2016

© Copyright by Jesse Harrison Laeuchli 2016

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

[illegible]

Jesse Harrison Laeuchli

Approved by the Committee, January 2016

[illegible]

Committee Chair
Professor Andreas Stathopoulos, Computer Science
The College of William and Mary

Weizhen Mao

Professor Weizhen Mao, Computer Science
The College of William and Mary

[illegible]

Professor Qun Li, Computer Science
The College of William and Mary

[illegible]

Associate Professor Robert Lewis, Computer Science
The College of William and Mary

[illegible]

Associate Professor Konstantinos Orginos, Physics
The College of William and Mary

ABSTRACT

Many applications such as path integral evaluation in Lattice Quantum Chromodynamics (LQCD), variance estimation of least square solutions and spline fits, and centrality measures in network analysis, require computing the diagonal of a function of a matrix, $\mathbf{Diag}(f(A))$ where A is sparse matrix, and f is some function. Unfortunately, when A is large, this can be computationally prohibitive. Because of this, many applications resort to Monte Carlo methods. However, Monte Carlo methods tend to converge slowly.

One method for dealing with this shortcoming is probing. Probing assumes that nodes that have a large distance between them in the graph of A, have only a small weight connection in $f(A)$. To determine the distances between nodes, probing forms A^k. Coloring the graph of this matrix will group nodes that have a high distance between them together, and thus a small connection in $f(A)$. This enables the construction of certain vectors, called probing vectors, that can capture the diagonals of $f(A)$. One drawback of probing is in many cases it is too expensive to compute and store A^k for the k that adequately determines which nodes have a strong connection in $f(A)$. Additionally, it is unlikely that the set of probing vectors required for A^k is a subset of the probing vectors needed for A^{k+1}. This means that if more accuracy in the estimation is required, all previously computed work must be discarded.

In the case where the underlying problem arises from a discretization of a partial differential equation (PDE) onto a lattice, we can make use of our knowledge of the geometry of the lattice to quickly create hierarchical colorings for the graph of A^k. A hierarchical coloring is one in which colors for A^{k+1} are created by splitting groups of nodes sharing a color in A^k. The hierarchical property ensures that the probing vectors used to estimate $\mathbf{Diag}(f(A))$ are nested subsets, so if the results are inaccurate the estimate can be improved without discarding the previous work.

If we do not have knowledge of the intrinsic geometry of the matrix, we propose two new classes of methods that improve on the results of probing. One method seeks to determine structural properties of the matrix $f(A)$ by obtaining random samples of the columns of $f(A)$. The other method leverages ideas arising from similar problems in graph partitioning, and makes use of the eigenvectors of $f(A)$ to form effective hierarchical colorings.

Our methods have thus far seen successful use in computational physics, where they have been applied to compute observables arising in LQCD. We hope that the refinements presented in this work will enable interesting applications in many other fields.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

It is difficult to convey my deep gratitude and respect for my advisor Andreas Stathopoulos, without whom this work would not have been possible. His keen perception, mathematical insight, and deep understanding of the field carried us though many difficult problems. To me he is the ideal computer scientist and mentor. I also wish to thank all of the members of my dissertation committee for their thoughtful comments and guidance, which greatly improved this work.

As part of the Computational Science research group at William and Mary, I had the good fortune to work with Lingfei Wu, and Eloy Romero Alcalde. I am grateful for their support and friendship.

During the course of this research I had several mentors at different internships, whose guidance and support contributed greatly to this work. In particular I would like to thank Chris Long, Lance Ward, and Geoff Sanders. They provided the spark for many ideas, and a great working environment to explore them.

For their many stimulating conversations on Computer Science, Mathematics and other topics, I would like to thank Philip Bozek, Douglas Tableman, and Walter McClean.

Finally, I would like to thank the Tan family for their moral support during the production of this thesis.

To My Parents, Samuel and Elizabeth Laeuchli

LIST OF TABLES

LIST OF FIGURES

Methods for Estimating the Diagonal of Matrix Functions

Chapter 1

Introduction

1.1 Motivation

In this work we study the problem of computing $\mathbf{Diag}(f(A))$, and $\sum_{i=1}^{N} \mathbf{Diag}(f(A))_i = \mathbf{Tr}(f(A))$, where A is a sparse matrix of size N and f is some function. Some useful examples are $f(A) = A^{-1}$, or $f(A) = \exp(A)$. When A is small, this can be computed directly, which is an $O(N^3)$ approach. When A is of intermediate size and is properly structured, recursive factorization methods allow for an $O(N^2)$ solution [70]. However, in many problems of interest, the size of A can be such that even $O(N^2)$ solutions are impractical. Because of this we abandon exact computation, and attempt to approximate the desired diagonals. There are two main methods for approximating the desired results. The first of these is based on Monte Carlo methods [45, 22]. The second is based on matrix sparsification [62], where we hope to ignore unimportant parts of the matrix in order to speed up the computation. Our work takes advantage of the features of both types of approximations in order to produce better algorithms.

The need to apply efficient solutions to this problem is a result of ever increasing matrix sizes in many diverse areas. Several examples are Statistics [45, 67], Lattice Quantum Chromodynamics (LQCD) [31], Material Science [28], and Network Analysis [68]. For example, in LQCD increasing the size of the lattice improves the physical accuracy of the simulation. Thus, there is significant interest in increasing the size of the lattices beyond what is currently computationally feasible. The same is true in the field of social network analysis. As social networks expand and represent ever more interconnected networks, performing analysis requires the use of increasingly large matrices. Finally, as large data sets become ever more prevalent, many statistical processes require matrices that continue to increase in size, necessitating faster methods.

1.1.1 Prior Work and New Approach

This problem of computing $\mathbf{Diag}(f(A))$ differs from common numerical analysis problems in a key way. Frequently, when similar problems are encountered, the problem is rewritten to be an optimization problem on a convex function. The problem can then be approached using optimization methods such as Newton's method, Gradient Descent, Conjugate Gradient and Non-Linear Conjugate Gradient, to converge to the value we are seeking. With this problem, no such optimization process can be undertaken. Because of this, statistical methods must be used.

Since this problem was first studied by Hutchinson [45], several statistical methods for the problem have been proposed. While these methods have the attractive feature that they provide statistical error estimates, they converge slowly and do not take advantage of information that the user may have about the matrix. The main goal of this work is to take advantage of anything that is known about a matrix in order to obtain a better estimate of $\mathbf{Diag}(f(A))$ in less time. Although the matrix $f(A)$ is normally dense, if the smallest elements of $f(A)$ are dropped, structure in this sparsified version of $f(A)$ will emerge. Since exactly obtaining this structure is no easier than solving the original problem, alternative information is used to approximate it. Once this structure is known, an estimate for $\mathbf{Diag}(f(A))$ can be obtained. This was the idea behind the method of [60] known as probing. Probing uses the powers of a matrix A to obtain an estimate of the structure of $f(A)$. For A^{-1} in particular this is based on the assumption that the Neumann series of A converges to $f(A)$. For different functions of A, other polynomials could be considered. Once an estimate for the structure of $f(A)$ is obtained, the authors of [62] show how to create probing vectors that allow for the recovery of the diagonal. However, this requires taking high powers of the matrix A, which is expensive to compute and may be impractical to store. Ideally, knowledge of the matrix A that is less expensive to obtain should be used. Further, the method in [60] provides no way to tell how accurate the error estimate is. Since this is important for many applications, this is a significant drawback.

Our proposed methods take advantage of several major areas of knowledge in order to obtain the structure of $f(A)$, in ways that are cheaper than finding powers of A. The first is geometric information. Many of our target applications arise from partial differential equations (PDE) that are discretized onto lattices. For a PDE given by $g(\mathbf{u}) = \mathbf{y}$, the solution at point $\mathbf{u}$ is often given in terms of the Green's function $G(\mathbf{u}, \mathbf{u'})$, where $\mathbf{u}$ is the point we are obtaining a solution for, and $\mathbf{u'}$ is some other point not equal to $\mathbf{u}$. For many PDEs, as $\|\mathbf{u} - \mathbf{u'}\|$ increases, $G(\mathbf{u}, \mathbf{u'})$ decays quickly since the physical forces the Green's function is attempting to model fall off rapidly with distance. Because of this, only connections between nodes that have short distances in the graph will have a large connection, or to put this another way, only elements in $f(A)$ corresponding to links between close nodes will be large. If we can determine which points have short distances between them in the graph, we can use this distance information to obtain an estimate for which are the large elements of $f(A)$. This is the previously mentioned approach of [60], where they use successively higher powers of A to compute the distance between nodes. In lattices, because of the known geometry, we can cheaply compute the distances between nodes without computing the powers of A.

For more general matrices where geometric information is lacking, we would prefer to bypass polynomial approximations which are expensive to compute, and work more directly with the structure of $f(A)$. This approach allows us to deal with matrices that do not exhibit the decay in interaction between distant nodes seen in many PDEs. We term this family of algorithms inverse probing. Inverse probing works by computing a subset of the columns v of the matrix $f(A)$. These columns can then be used by several algorithms to build an approximation to $f(A)$. The two main approaches we take are to form an approximation $vv^T \approx f(A)$, and to examine the values of v at different lags to try to predict the location of major off-diagonals of $f(A)$. This allows for the estimation of the magnitude of the connection between nodes directly.

A final method for obtaining information on the structure of $f(A)$ is based on examining the spectrum of $f(A)$. If probing were used to determine all the distance k connections of the i-th node, the algorithm would take the matrix vector product $A^k e_i$. However, this is similar to the process of obtaining a single iterate of the power iteration method, which would take the product $\frac{A^k r}{\|A^k r\|}$, where r is an appropriately chosen random vector. The power method is known to converge to the eigenvector of largest modulus of A. This suggests there could be a connection between probing and the largest eigenvector of A. The eigenvector holds the distance information that would be obtained by probing if it were taken an infinite number of steps. To state this slightly differently, the largest eigenvector holds similar structural information about $f(A)$ to that which probing could obtain. We present a heuristic that explores the connection between these two ideas and lays the basis for future research in this area.

The geometric, structural, and spectral information that our new algorithms make use of are all more computationally and storage efficient than the high powers of A required for probing. Additionally, in many cases we observe that they provide more accurate results then probing, because they obtain a more accurate representation of the structure of $f(A)$.

The final contribution of our work is to combine the idea of exploiting known but deterministic information about a matrix with statistical methods in order to provide the user with an improved as well as unbiased estimator. We provide a framework to analyze when the information obtained by our methods will provide meaningful improvements in the error estimates of our methods. By merging the strengths of the statistical and deterministic approaches we provide algorithms that are more robust.

1.2 Overview

The rest of this dissertation is structured as follows.

Chapter 2 We discuss in more detail the applications where $\mathbf{Diag}(f(A))$ is needed. We also examine the prior approaches, and determine the areas in which they are insufficient.

Chapter 3 We introduce a method for computing $\mathbf{Diag}(f(A))$ when A is a matrix arising from a toroidal lattice, that is, a lattice where the boundary conditions are periodic. Our method works by exploiting the geometry of the lattice. We show how the problem can be solved in the special case when the lattice has dimensions that are powers of 2, and in the more general case where the dimensions are arbitrarily sized. Finally we show how these methods can be combined with statistical approaches to provide unbiased estimators.

Chapter 4 We discuss the more general class of matrices, where the matrix is not a lattice but still has some structure that can be exploited. We examine structural and spectral types of information, and provide a framework for analyzing when enough structure exists for our algorithms to outperform previous approaches.

Conclusion We summarize our discussion.

Chapter 2

Prior Work and Applications

In this chapter we discuss related applications and prior work. Most prior applications are related to either computing $\mathbf{Tr}(A^{-1})$ or $\mathbf{Diag}(\exp(A))$. Given an $N \times N$ matrix A, with an eigendecomposition $A = VEV^{-1}$, the $\mathbf{Tr}(A^{-1}) = \sum_{i=1}^{N} \mathbf{Diag}(A^{-1})_i = \sum_{i=1}^{N} \frac{1}{E_{ii}}$. If the matrix is small, one could solve the problem directly by performing an LU decomposition [19], and then solving N linear equations for the diagonals of A^{-1}. Alternatively, one could obtain the eigenvalues of A, for example, by using the QR method [19], and summing them. Unfortunately, these approaches require $O(N^3)$ work, and so are impractical for the matrices that occur in the applications we are interested in.

The exponent of a matrix is given by $\exp(A) = V \exp(E) V^{-1} = \sum_{k=0}^{\infty} \frac{A^k}{k!}$. If the matrix is small enough, the eigendecomposition of the matrix can be computed, but in most cases of interest this is not practical. There are many methods proposed for forming $\exp(A)$ explicitly [69], but all of them require raising A to a high power. This can be computationally difficulty as well as requiring impractical amounts of storage, since in many cases A^k will become dense quickly.

2.1 Applications

2.1.1 Statistical Applications

We briefly consider applications of $\mathbf{Diag}(f(A))$ and $\mathbf{Tr}(f(A))$, to motivate our research. Such applications appear in physics, social network analysis, and statistics among others.

Computing $\mathbf{Diag}(f(A))$ shows up in several statistical problems. The simplest of these is computing the variance of a least squares problem [67]. In the case of a least squares problem, one would like to solve $min_x\|Bx - y\|$, where B is some arbitrarily sized matrix representing the observations we are trying to fit, that is likely singular. A solution can be found by computing the normal equation $x = A^{-1}z$, where $A = B^TB$ and $z = B^Ty$. While x can be solved using an iterative method, one would like to know what the variance of the solution is. It can be shown that given B, the covariance matrix of x is $(B^TB)^{-1}\sigma^2$, where σ^2 is the variance of the error of our fit. We do not have this variance available, but we can estimate it. Define $X_{n,p}$, as our n observations of p variables we are attempting to fit. Then as an estimator for σ^2 we have $\sigma^2 = \frac{1}{n-p}\|y - Bx\|^2 = \frac{1}{n-p}\sum_{i=1}^{n} e_i^2$, where e_i are the residuals $e_i = y_i - X_{i,1}z_1 - ... - X_{i,p}z_p$. The variance of the individual components z_i is then computed as $\mathbf{Diag}((B^TB))^{-1}{}_i\sigma_i^2$. Where B is large, representing many observations, inverting B^TB to obtain $\mathbf{Diag}(B^TB)$ is difficult.

Another area of statistics that this problem arises in, and which originally motivated Hutchinson [45] to develop his method, is fitting a spline to set of multidimensional noisy data $\mathbf{z}$ at irregularly spaced points $\mathbf{x}$. This is done by defining the function f, $\sum_{i=1}^{n}(z_i - f(x_i))^2 + \rho J(f)$, where n is the number of data points, and $J(f)$ is a rotation invariant measure of the roughness of f. This roughness is defined in terms of the partial derivates of f. The value ρ is a positive value controlling the degree of smoothing of the data, and is chosen to minimize the generalized cross validation function [71], which is defined as $GCV = \frac{\frac{1}{n}\|I-Az\|^2}{(\frac{1}{n}\mathbf{Tr}(I-A))^2}$, where A is the $n \times n$ symmetric influence matrix which takes the data values to their fitted values [72]. Forming this influence matrix requires inverting the spline matrix B. Because of this, the main expense of the validation is obtaining $\mathbf{Tr}(I - A)$.

2.1.2 Lattice Quantum Chromodynamics

Similar statistical issues arise in many areas of physics, and in paticular in Lattice Quantum Chromodynamics (LQCD) [31]. QCD is the theory of the behavior of the fundamental force known as the strong interaction, which describes the interactions among quarks, the building blocks of Hadrons. LQCD is a method for simulating these interactions. Since it is a non-perturbative method, it can be used to compute physical properties such as the masses of the various quarks, as well as the observables governing the coupling of the particles [31].

Unfortunately QCD gives rise to path integrals that are difficult to compute directly. If the system is discretized onto a 4D lattice, they can be approximated using Monte Carlo Integration. Normally this is done in two stages, by generating gauge fields according to a paticular probability distribution, then evaluating a correlation function that depends on these fields. The physical properties of interest are determined by a Monte Carlo average of the correleation functions generated by the ensemble of gauge fields [54].

This approach requires the computation of the trace of A^{-1}. Since the systems arising in this simulation are normally very large, most approaches in this area are based on iterative methods. Aside from the size of the matrix, they are also poorly conditioned, making their solution difficult. In particular, as the simulation parameters are tuned so that they more accurately represent the physical system of interest, the matrix starts to be become singular. Therefore, it is important to minimize the number of systems of equations that have to be solved to obtain an estimate for the trace, since each solution may take many iterations to converge. Despite these difficulties, LQCD has been a very successful approach, and our methods have wide applicability to it.

2.1.3 Network Centrality

Finally, we consider an application where the required $f(A)$ is not A^{-1}, but is instead $\exp(A)$,as in [68]. The authors are interested in computing the node centrality in a network, a metric of how important a particular node is in a given network. This question arises in social network analysis, as well as network design. In [68], the authors begin by defining a path as a list of distinct vertices connecting two nodes, and define a path that starts and stops at the same node as a closed path. They assume that nodes that have more closed paths are more important. Further, they give closed paths of differing lengths different weights, assigning to shorter paths a higher weight. If we let $k(i)_j$ be the number of paths of distance i for node j, and weight the paths with the inverse of their factorial distance, then we obtain the centrality metric $\sum_{i=0}^{\infty} \frac{k(i)_j}{i!}$. If one recalls that A^i_{jj}, which is the j-th diagonal element of the i-power of A, gives the number of round trip paths of length i for node j, then the desired equation is $\mathbf{Diag}(\sum_{i=0}^{\infty} \frac{A^i}{i!}) = \mathbf{Diag}(\exp(A))$.

2.2 Prior Work

2.2.1 Statistical Methods

Many of the applications shown in the prior section require extremely large matrices. Further, as computational resources expand, the applications will want to increase the size of the matrices in order to achieve more accurate results. This means that it is unlikely that attempts to solve this problem directly though matrix decomposition or eigensolvers, will ever be the best choice for most applications. Instead methods that attempt to statistically estimate it are needed.

The first attempt at such a statistical solution was made by Hutchinson [45], who was interested in calculating $\mathbf{Tr}(A^{-1})$ in order to compute splines. He showed that for a set of random vectors z_i, where each vector element is drawn independently from a Rademacher distribution, where each element has a $\frac{1}{2}$ chance to be 1 or -1,

$$\mathbf{Tr}(f(A)) = \mathbf{E}[z^T f(A) z] = \frac{\sum_{i=0}^{i=n} z_i^T f(A) z_i}{n}. \tag{2.1}$$

In his case, $f(A) = A^{-1}$. Taking advantage of the fact that $A^{-1}z = y$ can be solved by rewriting in the form $Ay = z$ and using an iterative solver, it is then possible to estimate the trace of A^{-1} even for very large matrices.

Following Hutchinson's work, the authors of [22] investigated several variations on Hutchinson's method, and proved bounds on their statistical variance, as well as on the number of samples needed to achieve a given accuracy, which can be seen in Table 2.1. Instead of taking z_i from (2.1) to be random vectors with elements from the Rademacher distribution, they examined the cases where the elements of z_i are Gaussian, where they are selected so that $z_i^H z_i = N$ which they term the Rayleigh-quotient estimator, and the case where the z_is are random unit vectors, $e_i = [0 \ldots 010 \ldots 0]$ where the 1 is in the i-th location. Additionally, they consider a variation on the scheme of taking z_i as unit vectors. Using these unit vectors directly computes a particular set of diagonal elements, and then attempts to extrapolate the missing diagonal elements from them. However, in cases where the values of the diagonal elements vary widely, this will work poorly. To counteract this, they instead compute $\frac{\sum_{i=0}^{n} z_i^T D^T A D z_i}{n}$, where D is either the Discrete Fourier Transform (DFT) matrix, or the Hadamard matrix. The DFT matrix is generated by $D = FFT(I)$, and the Hadamard matrix [73] is formed recurisvely, as

$$H_1 = [1], H_2 = \begin{bmatrix} 1, +1 \\ 1, -1 \end{bmatrix}, H_{2^k} = \begin{bmatrix} H_{2^{k-2}}, +H_{2^{k-1}} \\ H_{2^{k-2}}, -H_{2^{k-1}} \end{bmatrix} = H_2 \otimes H_{2^{k-1}}. \tag{2.2}$$

The Hadamard matrices have the disadvantage of only having sizes that are powers of two, but avoid the use of complex arithmetic, which is a requirement of using the DFT matrix. Since normally we will not need all N columns, we instead generate each matrix column by column as process which can be done efficiently [66].

Because these matrices are unitary, $D^T D = I$, $\mathbf{Tr}(D^T A D) = \mathbf{Tr}(D^T D A) = \mathbf{Tr}(IA) = \mathbf{Tr}(A)$, but has the effect of smoothing out the elements of the matrix A, thus making it less likely that an important diagonal element will be missed out by the estimator.

While [22] derives upper bounds for the number of vectors needed to achieve the probability of obtaining the desired amount of accuracy, these bounds are not tight. In practice the authors observe that the various methods perform almost identically. These

Table 2.1: Convergence rates of different methods.

Estimator	**Variance of the Sample**	**Bound on number of samples for an (ϵ,δ)-approx**	**Random bits per sample**
Gaussian	$2\|A\|_F$	$20\epsilon^{-2}ln(2/\delta)$	infinite;$\Theta(n)$ in floating point
Normalized Rayleigh-quotient	-	$\frac{1}{2}\epsilon^{-2}n^{-2}\mathrm{rank}^2(\mathrm{A})\ln(2/\delta)\mathrm{k}_\mathrm{f}^2(\mathrm{A})$	-
Hutchinson's	$2(\|A\|_F^2 - \sum_{i=1}^n A_{ii}^2)$	$6\epsilon^{-2}ln(2\mathrm{rank}(\mathrm{A})/\delta)$	$\theta(n)$
Unit Vector	$n\sum_{i=1}^n A_{ii}^2 - \mathbf{Tr}^2(A)$	$\frac{1}{2}\epsilon^{-2}ln(2/\delta)r_D^2(A)$, $r_D(A) = \frac{nmax_iA_{ii}}{\mathbf{Tr}(A)}$	$\theta(logn)$
Mixed Unit Vector(DFT/Hadamard)	-	$8\epsilon^{-2}ln(4n^2/\delta)ln(4/\delta)$	$\theta(logn)$

methods converge slowly and cannot be improved without additional information about the matrix. However, it is seldom the case that no useful knowledge of the matrix is available or that cannot be extracted by approximation techniques.

2.2.2 Non-Statistical Methods

Several deterministic methods have been proposed that solve this problem exactly, in the case where $f(A) = A^{-1}$, by preforming some form of matrix factorization, and avoid the problem of slow convergence that the statistical methods have. These approaches have serious drawbacks however. In [49], the authors introduce a method which works by finding a hierarchy of Schur complements of matrices arising from grids, but the run time of this method is of order $O(N^{3/2})$ and $O(N^2)$ for the 2D and 3D case respectively. Thus for sufficiently large N, or for higher dimensional problems, this approach is not practical. Further it does not address the case of matrices that do not arise from PDEs.

The method introduced in [70] performs an LU factorization and computes the last diagonal entry of the inverse directly from this factorization. It then reorders the nodes so that each diagonal is in turn the last element of the LU matrix. In order to avoid computing a unique LU factorization for every reordering, they decompose each LU factorization into partial LU factorizations. This method has similar drawbacks to those in [49], requiring that the matrix arise from a PDE and has a run time of $O(N^2)$ for a 2D matrix.

Proposed in [67] is a method based on Takahashi's equations, which allow a subset of the elements of A^{-1} to be recursively computed, using only the elements of an LDU decomposition of A, and the previously generated elements of A^{-1}, with the first step of the recursive process requiring only the elements of the LDU decomposition to compute.

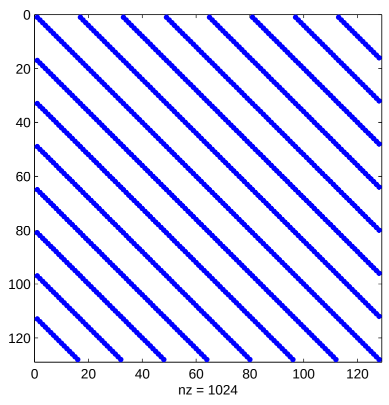
The subset of elements which can be computed in this manner are those elements that are non-zero in the LDU decomposition. Given a sparse LDU decomposition, it follows that the number of elements needed to compute $\mathbf{Diag}(A^{-1})$ is small. However, while the subset of elements A^{-1} needed to compute $\mathbf{Diag}(A^{-1})$ may be smaller than that needed to compute the entire matrix A^{-1}, it can still be quite large.

Alternative approaches have been developed that avoid computing the result directly, which is infeasible for large problems, while still making use of any information that is known about the problem. The main idea behind them is to create the vectors in (2.1) in such a way that any available structure is exploited. This method was first introduced in [28] . The main insight is that many matrices in practice have an inverse with a periodic and decaying structure where the magnitude of the elements falls off away from the main diagonal of the matrix. Therefore we can set the z_is to be such that they zero out as many diagonals of the matrix as possible. If the contribution to the error from the diagonals that have not been zeroed out is small, the results will be very accurate. As more z_is are used, more diagonals are zeroed out, and the solution becomes more accurate. Further, we show later how, if this is paired with statistical methods, this will reduce the variance, because there will be fewer elements contributing to the sums in Table 2.1. To achieve this zeroing out effect, the vectors of the Hadamard matrix are used in their natural order, since they zero all contributions to the error except those elements from an increasingly small subset of diagonals, as can be seen in Figure 2.1.

Another useful method supplied in [28] is how to calculate the diagonal of A instead of simply the trace. They show that the following estimator will converge to the diagonal

$$\mathbf{Diag}(A) \approx (\sum_{i=1}^{n} z_i \odot Az_i) \oslash (\sum_{i=1}^{n} z_i \odot z_i) \tag{2.3}$$

where $\odot$ is componentwise multiplication and $\oslash$ is componentwise division, and the z_is are random vectors. This has the same drawback as (2.1), in that while in expectation this will yield the correct answer, in practice convergence can be very slow. Therefore the


(a) 16 Hadamard vectors

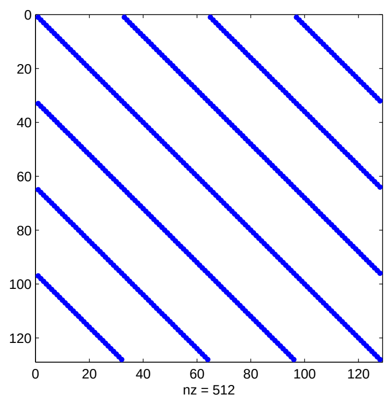

(b) 32 Hadamard vectors

Figure 2.1: The area zeroed out by using Hadamard vectors. As the number of vectors increases, the number of diagonals that contribute to the error decease, and become further from the main diagonal.

question of picking the z_is to exploit the structure of A is the same as for estimating the trace.

While the Hadamard based method of [28] works well for a specific class of matrices which are generally those arising from a PDE with a Green's function describing a force that decays with distance, matrices without this diagonal structure do not benefit as much. An attempt to exploit less regularly ordered structure is behind the idea of probing. Probing has been a useful technique with a long history in the context of approximating the Jacobian matrix [17, 38], or other matrices [18]. Its use for approximating the diagonal of A^{-1} was proposed in [60] because it finds the most important areas of A^{-1} rather than the fixed structure removed by the Hadamard approach. Probing recovers the diagonals of a sparse matrix by finding the coloring of its associated graph. Coloring a graph involves assigning a color to each vertex in such a way that no two connected vertices share a color. Unfortunately, finding the optimal coloring in the sense of using the least number of colors is an NP-Complete problem. However, for many graphs a greedy algorithm performs well [61], and is the approach used in probing. When the rows and columns of a matrix are arranged so that all nodes that share the same color are adjacent, a zero block diagonal structure will result, as can be seen in Figure 2.2. This structure is due to the fact that since these nodes share a color, they must have no connection (otherwise this would be an invalid coloring).

This block-diagonal zero structure can be exploited to recover the diagonal of the matrix by creating probing vectors. Given a coloring C for A, with c total colors, we will need only c vectors to recover the diagonal. We generate a probing vector for each color m, and set the i-th element of that vector to be 1 if the i-th node of the graph of A was assigned the m-th color and zero elsewhere as seen in (2.4). These vectors can then be used with (2.1) or (2.3).

$$p_i^m = \begin{cases} 1, & \text{if } i \in C_m \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

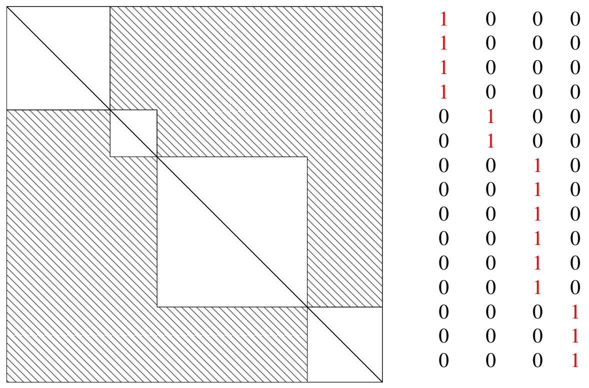


Figure 2.2: Probing a four colorable graph. The diagonal elements of the graph can be recovered using the probing vectors shown.

Unfortunately, $f(A)$ is normally dense. Because of this the associated graph of $f(A)$ is fully connected, and every node will be assigned a unique color. To avoid this probing, structure for $f(A)$ must be induced by sparsification. If the smallest magnitude elements of $f(A)$ are dropped, then $f(A)$ will appear, which can be exploited by probing. Of course, computing $f(A)$ and then sparsifying it, is not easier than the original problem of obtaining **Diag**$(f(A))$. To counteract this issue, [62] introduced the idea of probing using a matrix polynomial $q_n(A)$. They find a polynomial of matrix A, such that $q_n(A) \approx A^{-1}$, as the order n of the polynomial increases. In their paper they use the Neumann approximation to A^{-1}, which is $A^{-1} \approx (\sum_{j=0}^{n} M^{-1}Q^{j})M^{-1}$, where $A = M - Q$, and $M = \textbf{Diag}(A)$. However, they are interested only in the structure, and not the values of $q_n(A)$. Since in the case of the Neumann approximation the nonzero structure of $q_n(A)$ is a subset of the nonzero structure of $q_{n+1}(A)$, they need only color A^n to obtain an approximation to the structure of A^{-1} after sparsification. This method has the drawback that $q_n(A)$ quickly becomes denser and thus expensive to compute for even small n, when A is large.

An additional drawback with probing is that once the coloring for a particular $q_n(A)$ is obtained, and probing vectors are created, it is possible that the estimate it produces will not be sufficiently accurate. However, it is unlikely that the probing vectors of $q_n(A)$ will be a subset of those created for $q_m(A)$, $m < n$. In this case, all the previous work computing $f(A)z$ will have to be discarded. Since these results are produced by solving large linear systems iteratively, this is a serious shortcoming. An example illustrating this may be seen in Figure 2.3. In the first case, the associated graph of $q_1(A)$, the associated graph was colored with three colors, but required four colors to color the graph of $q_2(A)$. Unfortunately the resultant probing vectors do not span the original probing vectors. If on the other hand the colors had split as seen in the second example, the initial probing vectors would have been spanned by the new ones, meaning the work of computing $f(A)z$ need not have gone to waste.

$$\begin{matrix}1&0&0\\1&0&0\\0&1&0\\0&1&0\\0&0&1\\0&0&1\end{matrix} \not\subset \begin{matrix}1&0&0&0\\0&1&0&0\\0&1&0&0\\0&0&1&0\\0&0&1&0\\0&0&0&1\end{matrix} \quad \textbf{but} \quad \begin{matrix}1&0&0\\1&0&0\\0&1&0\\0&1&0\\0&0&1\\0&0&1\end{matrix} \subset \begin{matrix}1&0&0&0\\1&0&0&0\\0&1&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{matrix}$$

Figure 2.3: An example of wasted vectors in probing, versus an example where the vectors can be reused.

Chapter 3

Estimation of Diag$(f(A))$ on torodial lattices

In this chapter, we discuss how to compute $\mathbf{Diag}(f(A))$ in the case where A represents a toroidal lattice. In this case the geometric information can be leveraged to provide high quality probing vectors very quickly. In addition, these vectors can be constructed in such a way that if the accuracy of the result is insufficient, the process can continue by reusing our previously computed results, a process we term Hierarchical Probing. While our solution works for lattices of arbitrary dimensions, we also present an even faster method for lattices which have dimensions of powers of two.

3.1 Lattices with dimensions consisting only of powers of 2

3.1.1 Introduction

Two methods that have previously attempted to address this problem by exploiting the structure of the matrix A are the approach of using Hadamard vectors [28], and the method of probing [62]. We combine ideas from both of these methods in order to overcome their respective shortcomings.

The approach based on the use of Hadamard vectors discussed in Section 2.2.2 borrows ideas from coding theory and selects deterministic vectors for the Monte Carlo(MC) estimate given in 2.1 as columns of a Hadamard matrix [28]. These vectors are orthogonal and, although they produce the exact answer in N steps, their benefit stems from systematically capturing certain diagonals of the matrix, as in figure 2.1. For example, if we use the first 2^m Hadamard vectors, the error in the trace approximation comes only from non-zero elements on the $(k2^m)$th matrix diagonal, $k = 1, \dots, N/2^m$. Thus, the MC iteration continues annihilating more diagonals with more Hadamard vectors, until it achieves the required accuracy. However, in most practical problems the matrix bandwidth is too large, the non-zero diagonals do not fall on the required positions, or the matrix is not even sparse (which is typically the case for A^{-1}).

Probing [62] attempts to select vectors that annihilate the error contribution from the heaviest elements of A^{-1}. For a large class of sparse matrices, elements of A^{-1} decay exponentially away from the non-zero structure of A. By this we mean that the magnitude of the $A^{-1}_{i,j}$ element relates to the distance of the shortest path between nodes i and j in the graph of A. Assume that the graph of A has a distance-k coloring (or distance-1 coloring of the graph of A^k) with m colors. Then, if we define the vectors $z_j, j = 1, \dots, m$, with $z_j(i) = 1$ if color(i)=j, and $z_j(i) = 0$ otherwise, we obtain $\mathbf{Tr}(A) = \sum_{j=1}^{m} z_j^T A z_j$. For $\mathbf{Tr}(A^{-1})$ the equation is not exact, but it does not include errors from all elements of A^{-1} that correspond to paths between vertices that are distance-k neighbors in A. The probing technique has been used for decades in the context of approximating the Jacobian matrix [35, 38] or other matrices [57]. Its use for approximating the diagonal of A^{-1} in [62] (see also [28]) is promising as it selects the important areas of A^{-1} rather than the predetermined structure dictated by Hadamard vectors. However, the accuracy of the trace estimate obtained through a specific distance-k probing can only be improved by applying Monte Carlo, using random vectors that follow the structure of each probing vector. To take advantage of a higher distance probing, all previous work has to be discarded, and the method rerun for a larger k. We discuss this in Sections 3.1.5 and 3.1.7.

We introduce hierarchical probing which avoids the problems of the previous two methods. It annihilates error stemming from the heaviest parts of A^{-1}, and it does so incrementally, until the required accuracy is met. To achieve this, we relax the requirement of distance-k coloring of the entire graph. The idea is to obtain recursively a (suboptimal) distance-2^{i+1} coloring by independently computing distance-1 colorings of the subgraphs corresponding to each color from the distance-2^i coloring. The recursion stops when all the color-subgraphs are dense, i.e., we have covered all distances up to the diameter of the graph. We call this method, "hierarchical coloring". For regular, toroidal lattices each subgroup has the same number of colors, which enables an elegant, hierarchical basis for probing based on an appropriate ordering of the Hadamard and/or Fourier vectors: the first m such vectors constitute a basis for the corresponding m probing vectors. We call this method, "hierarchical probing". It can be implemented using only bit arithmetic, independently on each lattice site. We also address the issue of statistical bias by viewing hierarchical probing as a method to create a hierarchical basis starting from any vector, including random.

SubSection 3.2.2 presents some background for these methods and describes the limitations of classical probing. In subSection 3.2.3, we introduce the idea of hierarchical coloring and, for the case of uniform grids and tori, we develop a hierarchical coloring method that uses only local coordinate information and bit operations. In subSection 3.2.4, we use this coloring to produce a sequence of hierarchical probing vectors. In subSection 3.2.5, we provide several experiments for typical lattices and problems from LQCD that show that MC with hierarchical probing has much smaller variance than random vectors and performs equally well or better than the expensive, classical probing method.

3.1.2 Preliminaries

We use vector subscripts to denote the order of a sequence of vectors, and parentheses to denote the index of the entries of a vector. We use MATLAB notation to refer to row or column numbers and ranges. The matrix A, of size $N \times N$, is assumed to have a symmetric structure (undirected graph).

3.1.3 Lattice QCD problems

Lattice Quantum ChromoDynamics (LQCD) is a formulation of Quantum Chromo-Dynamics (QCD) that allows for numerical calculations of properties of strongly interacting matter (Hadron Physics) [64]. These calculations are performed through Monte Carlo computations of the discretized theory on a finite 4 dimensional Euclidean lattice. Physical results are obtained after extrapolation of the lattice spacing to zero. Hence calculations on multiple lattice sizes are required for taking the continuum and infinite volume limits. In this formulation, a large sparse matrix D called the Dirac matrix plays a central role. This matrix depends explicitly on the gauge fields U. The physical observables in a LQCD calculation are computed as averages over the ensemble of gauge field configurations. In various stages of the computation one needs, among other things, to estimate the determinant as well as the trace of the inverse of this matrix. The dimensionality of the matrix is $3 \times 4 \times L_s^3 \times L_t$, where L_s and L_t are the dimensions of the spatial and temporal directions of the space-time lattice, 3 is the dimension of an internal space named "color", and 4 is the dimension of the space associated with the spin and particle/antiparticle degrees of freedom. Typical lattice sizes in todays calculations have $L_s = 32$ and $L_t = 64$ and the largest calculations performed on leadership class machines at DOE or NSF supercomputing centers have $L_s = 64$ and $L_t = 128$. As computational resources increase and precision requirements grow, lattices will become even bigger.

3.1.4 The Monte Carlo method for $\mathbf{Tr}(A^{-1})$

Hutchinson introduced the standard MC method for estimating the trace of A and proved the following [45].

Lemma 3.1 *Let A be a matrix of size $N \times N$ and denote by $\tilde{A} = A - \mathbf{Diag}(A)$. Let z be a $\mathbb{Z}_2$ random vector (i.e., whose entries are i.i.d Rademacher random variables $Pr(z(i) =$*

$\pm 1) = 1/2$*). Then,* $z^T Az$ *is an unbiased estimator of* $\mathbf{Tr}(A)$*, i.e.,*

$$E(z^T Az) = \mathbf{Tr}(A),$$

and

$$var(z^T Az) = \|\tilde{A}\|_F^2 = 2\left(\|A\|_F^2 - \sum_{i=1}^{N} A(i,i)^2\right).$$

The MC method converges with rate $\sqrt{var(z^T Az)/s}$, where s is the sample size of the estimator (number of random vectors). Thus, the MC converges in one step for diagonal matrices, and very fast for strongly diagonal dominant matrices. More relevant to our $\mathbf{Tr}(A^{-1})$ problem is that large off-diagonal elements of A^{-1} contribute more to the variance $\|\widetilde{A^{-1}}\|_F^2$ and thus to slower convergence.

Computationally, $z^T A^{-1} z$ (often regarded as a Gaussian quadrature) can be computed using the Lanczos method [24, 39, 58]. This method also produces upper and lower bounds on the quadrature, which are useful for terminating the process. The alternative of solving the linear system $A^{-1}z$, is not recommended for non-Hermitian systems because of worse floating point behavior [59], but for Hermitian systems it can be as effective if we stop the system earlier. Specifically, the quadrature error in Lanczos converges as the square of the system residual norm [39], and therefore we need only let the residual converge to the square root of the required tolerance. A potential advantage of solving $A^{-1}z$ is that the result can be reused when computing multiple correlation functions involving bilinear forms $y^T A^{-1} z$ (e.g., in LQCD).

3.1.5 Probing

Probing has been used extensively for the estimation of sparse Jacobians [35, 38], for preconditioning [57], and in Density Functional Theory for approximating the diagonal of a dense projector whose elements decay away from the main diagonal [28, 62]. The idea is to expose the structure and recover the non-zero entries of a matrix by multiplying it with a small, specially chosen set of vectors. For example, we can recover the elements of a diagonal matrix through a matrix-vector multiplication with the vector of N 1's, $\mathbf{1}_N = [1, \ldots, 1]^T$. Similarly, a banded matrix of bandwidth b can be found by matrix-vector multiplications with vectors $z_k, k = 1, \ldots, b$, where

$$z_k(i) = \begin{cases} 1, & \text{for } i = k : b : N \\ 0, & \text{otherwise} \end{cases} .$$

To find the trace (or more generally the main diagonal) of a matrix, the methods are based on the following proposition [28].

Proposition 3.1 *Let $Z \in \Re^{N \times s}$ be the matrix of the s vectors used in the MC trace estimator. If the i-th row of Z is orthogonal to all those rows j of Z for which $A(i,j) \neq 0$, then the trace estimator yields the exact* $\mathbf{Tr}(A)$.

In the above example of a banded matrix, we choose the vectors z_k such that their rows only overlap for structurally orthogonal rows of A (i.e., for rows farther than b apart). Thus, by proposition 3.1, the trace computed with these z_k is exact.

If A is not banded but its sparsity pattern is known, graph coloring can be used to identify the structurally orthogonal segments of rows, and derive the appropriate probing vectors [62]. Assume the graph of A is colorable with m colors, each color having $n(k)$ number of vertices, $k = 1, \ldots, m$. The coloring is best visualized by letting q be the permutation vector which reorders identically colored vertices together. Then $A(q,q)$ has m blocks along the diagonal, the k-th block is of dimension $n(k)$, and each block is a diagonal matrix. Figure 3.1 shows an example of the sparsity structure of a permuted 4-colorable matrix. Computationally, permuting A is not needed. If we define the vectors:

$$z_k(i) = \begin{cases} 1 & \text{if } color(i) = k \\ 0 & \text{otherwise} \end{cases} , k = 1, \ldots, m \tag{3.1}$$

we see that Proposition 3.1 applies, and therefore $\mathbf{Tr}(A) = \sum_{k=1}^{m} z_k^T A z_k$.

When the matrix is dense and all its elements are of similar magnitude, there is no structure to be exploited by probing. The inverse of a sparse matrix is typically dense,

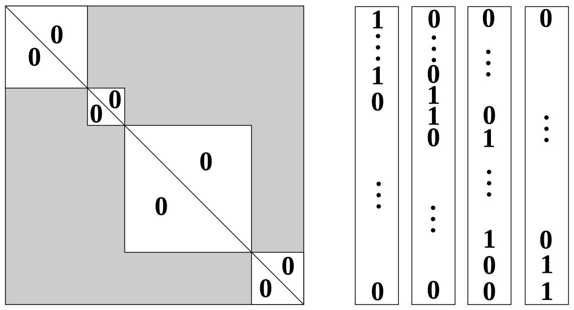


Figure 3.1: Visualizing a 4-colorable matrix permuted such that all rows corresponding to color 1 appear first, for color 2 appear second, and so on. Each diagonal block is a diagonal matrix. The four probing vectors with 1s in the corresponding blocks are shown on the right.

but, for many applications, its elements decay on locations that are farther from the locations of the non-zero elements of A. Such small elements of A^{-1} can be dropped, and the remaining A^{-1} is sparse and thus colorable. Diagonal dominance of the matrix is a sufficient (but not necessary) condition for the decay to occur [35, 62]. This property is exploited by approximate inverse preconditioners and can be explained from various points of view, including Green's function for differential operators, the power series expansion of A^{-1}, or a purely graph theoretical view [29, 30, 34, 44]. In the context of probing, we drop elements $A^{-1}(i, j)$ where the vertices i and j are farther than k links apart in the graph of A. Because this graph corresponds to the matrix A^k, our required distance-k coloring is simply the distance-1 coloring of the matrix A^k [38, 62]. Computing A^k for large k, however, is time and/or memory intensive.

The effectiveness of probing depends on the decay properties of the elements of A^{-1}, and the choice of k in the distance-k coloring. The problem is that k depends both on the structure and the numerical properties of the matrix. If elements of A^{-1} exhibit slow decay, choosing k too small does not produce sufficiently accurate estimates because large elements of A^{-1} (linking vertices that are farther than k apart) contribute to the variance in Lemma 3.1. Choosing k too large increases the number of quadratures unnecessarily, and more importantly, makes the coloring of A^k prohibitive. This problem has also been identified in [62] but no solution proposed.

A conservative approach is to use probing for a small distance (typically 1 or 2) to remove the variance associated only with the largest, off-diagonal parts of the matrix. Then, for each of the resulting m probing vectors, we generate s random vectors that follow the non-zero structure of the corresponding probing vector, and perform s MC steps (requiring ms quadratures). In LQCD this method is called dilution. In its most common form it performs a 2-color (red-black) ordering on the uniform lattice and uses the MC estimator to compute two partial traces: one restricted on the red sites, the other on the black sites of the lattice [25, 37, 51]. Therefore, all variance caused by the direct red-black connections of A^{-1} is removed. The improvement is modest, however, so additional "dilution" is required [23, 51].

3.1.6 Hadamard vectors

An $N \times N$ matrix H is a Hadamard matrix of order N if it has entries $H(i,j) = \pm 1$ and $HH^T = NI$, where I is the identity matrix of order N [28, 43]. N must be 1, 2, or a multiple of 4. We restrict our attention to Hadamard matrices whose order is a power of 2, and can be recursively obtained as:

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad H_{2n} = \begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix} = H_2 \otimes H_n.$$

For powers of two, H_n is also symmetric, and its elements can be obtained directly as

$$H_n(i,j) = (-1)^{\sum_{k=1}^{\log N} i_k j_k}, \tag{3.2}$$

where $(i_{\log N}, \ldots, i_1)_2$ and $(j_{\log N}, \ldots, j_1)_2$ are the binary representations of of $i-1$ and $j-1$ respectively. We also use the following notation to denote Hadamard columns (vectors): $h_j = H_n(:, j+1), j = 0, \ldots, n-1$. Hadamard matrices are often called the integer version of the discrete Fourier matrices,

$$F_n(j,k) = e^{2\pi(j-1)(k-1)\sqrt{-1}/n}. \tag{3.3}$$

For $n = 2$, $H_2 = F_2$, but for $n > 2$, F_n are complex. These matrices have been studied extensively in coding theory where the problem is to design a code (a set of $s < N$ vectors Z) for which ZZ^T is as close to identity as possible [28]. H_n and F_n vectors satisfy the well known Welch bounds but H_n vectors do not achieve equality [63]. Moreover, F_n are not restricted to powers of two. Still, Hadamard matrices involve only real arithmetic, which is important for efficiency and interoperability with real codes, and it is easy to identify the non-zero pattern they generate. Later, we will view the Hadamard matrix as a methodical way to build an orthogonal basis.

Consider the first 2^k columns of a Hadamard matrix $Z = H(:, 1 : 2^k)$. The non-zero pattern of the matrix ZZ^T consists of the $i2^k$ upper and lower diagonals, $i = 0, 1, \ldots$ [28]. Because $\mathbf{Tr}(Z^T A^{-1} Z) = \mathbf{Tr}(A^{-1} ZZ^T)$ and because of Lemma 3.1 and Proposition 3.1, the error in the MC estimation of the trace is induced only by the off-diagonal elements of A^{-1} that appear on the same locations as the non-zero diagonals of ZZ^T. If the matrix is banded or its diagonals do not coincide with the ones of ZZ^T, the trace estimation is exact. When the off-diagonal elements of A^{-1} decay exponentially away from the main diagonal, increasing the number of Hadamard vectors achieves a consistent (if not monotonic) reduction of the error. We note that this special structure of ZZ^T is achieved only when the number of vectors, s, is a power of two. For $2^k < s < 2^{k+1}$, the structure of ZZ^T is dense in general, but the weight of ZZ^T elements is largest on the main diagonal (equal to s) and decreases between diagonals $i2^k$ and $(i+1)2^k$. Thus, estimation accuracy improves with s, even for dense matrices. However, to annihilate a certain sparsity structure of a matrix, the estimates at only $s = 2^k$ should be considered. Similar properties apply for the F_n matrices.

3.1.7 Overcoming probing limitations

We seek to construct special vectors for the MC estimator that perform at least as well as $\mathbb{Z}_2$ noise vectors, but can also exploit the structure of the matrix, when such structure exists. Although Hadamard vectors seem natural for banded matrices, they cannot handle

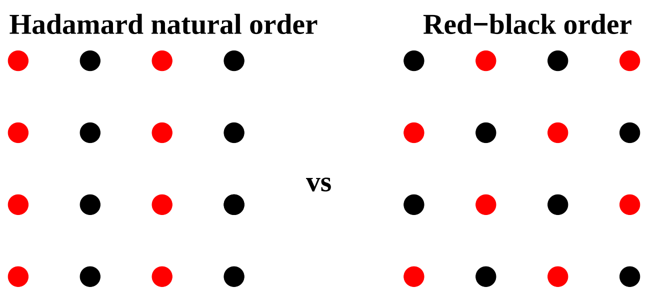


Figure 3.2: Crossed out nodes have their contribution to the error canceled by the Hadamard vectors used. Left: the first two, natural order Hadamard vectors do not cancel errors in some distance-1 neighbors in the lexicographic ordering of a 2-D uniform lattice. Right: if the grid is permuted with the red nodes first, the first and the middle Hadamard vectors completely cancel variance from nearest neighbors and correspond to the distance-1 probing vectors.

deviations from this structure. For example, the first two Hadamard vectors compute the exact trace of a tridiagonal matrix. For the matrix that corresponds to a 2-D uniform lattice of size $2^n \times 2^n$ with periodic boundary conditions and lexicographic ordering, producing the exact trace requires the first 2^{n+1} Hadamard vectors. However, if we consider the red-black ordering of the same matrix, only two Hadamard vectors, h_0 and $h_{2^{n-1}}$, are sufficient, as shown in Figure 3.2.

The previous example shows that although Hadamard vectors are a useful tool, probing is the method that discovers matrix structure. Therefore, we turn to the problem of how to perform probing efficiently on A^k and for large k. Ideally, a method should start with a small k and increase it until it achieves sufficient accuracy. However, the colorings, and thus the probing vectors, for two different k's are not related in general. Thus, in addition to the expense of the new coloring, all previously performed quadratures must be discarded.

First let us persuade the reader that work from a previous distance-k probing cannot be reused in general. Assume the distance-1 coloring of a matrix of size 6 produced three colors: color 1 has rows 1 and 2, color 2 has rows 3 and 4, color 3 has rows 5 and 6. Next we perform a distance-2 coloring of A, and assume there are four colors: color 1 has row 1, color 2 has rows 2 and 3, color 3 has rows 4 and 5, color 4 has row 6. As in Figure 3.1, the distance-1 and distance-2 probing vectors, $Z^{(1)}$ and $Z^{(2)}$ respectively, are the following:

$$Z^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \; Z^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Unfortunately, the three computed quadratures $Z^{(1)T}A^{-1}Z^{(1)}$ (or solutions to $A^{-1}Z^{(1)}$) cannot be used to avoid recomputation of the four quadratures $Z^{(2)T}A^{-1}Z^{(2)}$.

Consider now a matrix of size 8 with two colors in its distance-1 coloring. Assume that its distance-2 coloring produces four colors, and that all rows with the same color belong also to the same color group for distance-1. Then the subspace of the corresponding probing vectors is spanned by certain Hadamard vectors:

$$Z^{(1)} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \in span(\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix}), \quad Z^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \in span(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix}).$$

The four Hadamard vectors are h_0, h_4, h_2, h_6. More interesting than the equality of the spans is that the two bases are an orthogonal transformation of each other. Specifically, $Z^{(1)} = 1/2[h_0, h_4]H_2$ and $Z^{(2)} = 1/2[h_0, h_4, h_2, h_6]H_4$. Because the trace is invariant under orthogonal transformations, we can use the Hadamard vectors instead (as we implicitly did in Figure 3.2 for the lattice). Clearly, for this case, the quadratures of the first two vectors can be reused so that the distance-2 probing will need computations for only two additional vectors.

A key difference between the two examples is the nesting of colors between successive colorings. In general, such nesting cannot be expected and thus an incremental approach to probing will necessarily discard prior work. A second difference is that all color groups are split into the same number of colors in the successive coloring. To achieve these desired characteristics, we develop first a hierarchical, all-distance coloring, and then represent its probing basis through a convenient set of vectors. We explain the general hierarchical coloring idea next.

3.1.8 Hierarchical coloring

The key idea is to enforce nesting of colors in successive distance colorings by looking at each color group independently and coloring its nodes for the next higher distance using only local information from that color group. We apply this recursively until every node of the graph is colored uniquely.

We begin this recursive approach by finding a distance-1 coloring of the graph, thus partitioning its nodes into separate color groups. In subsequent levels of the hierarchy, the nodes in each group will never share a color with the nodes in another group. To move to the next level, we use the graph at the current level to produce a distance-2 connectivity among the nodes within each color group. Then we apply the algorithm recursively to the induced subgraphs for each color group independently. It is a straightforward inductive observation that at every level we generate a distance-2^i coloring, but for all $i = 0, 1, \ldots$, until the distance covers the diameter of the graph.

Hierarchical coloring produces more colors at distance-2^i than classical coloring of the graph of A^{2^i}. If the task were to approximate the trace of the matrix A^{2^i}, the extra colors would be redundant and the additional probing vectors would represent unnecessary work. However, we approximate the trace of A^{-1}, which is dense. Thus, the larger number of hierarchical probing vectors at distance-2^i will also approximate some elements that represent node distances larger than 2^i, yielding a larger variance reduction than the corresponding A^{2^i} classical probing method.

3.1.9 Hierarchical coloring on lattices

Uniform d-D lattices allow for an extremely efficient implementation of the hierarchical coloring approach, based entirely on bit-arithmetic.

Consider first the 1-D case, where the lattice has $N = 2^k$ points, where $k = \log_2 N$, which guarantees the 2-colorability of the 1-D torus. Any point has a coordinate $0 \leq x \leq N-1$ with a binary representation: $[b_k, b_{k-1}, \ldots, b_1] = \text{dec2bin}(x)$. At the first level, the distance-1 coloring is simply red-black (we associate red with 0 and black with 1), and x gets the color of its least significant bit (LSB), b_1. In the coloring permutation, we order first the $N/2$ red nodes. At the second level, we consider red and black points separately and split each color again, but now based on the second bit b_2. Thus, points $[**\ldots**00]$ and $[**\ldots**10]$ take different colors, and by construction all colors are given hierarchically. The second level permutation will not mix nodes between the first two halves of the first level, but will permute nodes within the respective halves, i.e., points with 0 in the LSB always appear in the first half of the permutation. The process is repeated recursively for each color, until all points have a different color.

The binary tree built by the recursive algorithm splits the points of a subtree in half at the i-th level based on b_i. Thus, to find the final permutation we trace the path from the root to a leaf, producing the binary string: $[b_1 b_2 \ldots b_k]$, which is the bit reversed string for x. Denote by P the final permutation array such that node $x = 0, \ldots, N-1$ in the original ordering is found in location $P(x)$ of the final permutation. Then,

$$P(x) = \text{bin2dec}(\text{bitreverse}(\text{dec2}bin(x))) \tag{3.4}$$

and the computation is completely independent for any coordinate.

Extending to torus lattices of d dimensions, where $N = \prod_{j=1}^{d} 2^{k_j}$, has three complications: First, the subgraph of the same color nodes is not a conformal uniform lattice. Second, the geometry does not allow a simple bit reversal algorithm. Third, not all dimensions have the same size ($k_j \neq k_i$). The following sections address these.

3.1.10 Splitting color blocks into conformal d-D lattices

Consider a point with d coordinates $(x_1, x_2, \ldots, x_d)$. Let $[b^j_{k_j}, \ldots, b^j_2, b^j_1]$ be the binary representation of coordinate x_j with $0 \le x_j < 2^{k_j}$. We know that uniform lattices are 2-colorable, so at the first level, red black ordering involves the least significant bit of all coordinates. The color assigned to the point is $\mathrm{mod}(\sum_{j=1}^{d} b^j_1, 2)$. However, the red partition, which is half of the lattice points, is not a regular d-dimensional torus. Every red point is distance-2 away from any red neighbor, and therefore it has more neighbors (e.g., in case of 2-D it is connected with 8 neighbors, in 3-D with 18, and so on). To facilitate a recursive approach, we observe that the reds can be split into 2^{d-1} d-dimensional sublattices, if we consider them in groups of every other row in each dimension. Similarly for the blacks. For the 2-D case this is shown in Figure 3.3.

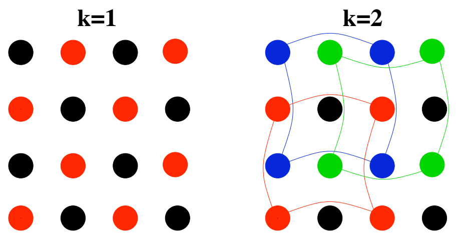


Figure 3.3: When doubling the probing distance (here from 1 to 2) we first split the 2-D grid to four conformal 2-D subgrids. Red nodes split to two 2×2 grids (red and green), and similarly black nodes split to blues and black. Smaller 2-D grids can then be red-black ordered.

This partitioning is obtained based on the value of the binary string: $[b^1_1, b^2_1, \ldots, b^d_1]$. For each value, the resulting sublattice contains all points with the given least significant bits in its d coordinates. Because each coordinate loses one bit, the size of each sublattice is $\prod_{j=1}^{d} 2^{k_j-1}$. At this second level, each of the 2^d sublattices can be 2-colored independently. Each sublattice will receive a distinct coloring, which which will be hierarchical, as long as we track the original color of the nodes.

3.1.11 Facilitating bit reversal in higher dimensions

The above splitting based on the LSBs from the d coordinates does not order the adjacent colors together. For example, the partitioning at the first level of $d = 2$ gives four sub-lattices (00,01,10,11) of which the 00 and 11 are reds while 01 and 10 are blacks. We can recursively continue partitioning and coloring the sublattices. However, if we concatenate at every level the new 2 bits from the 2 coordinates, as in the bit reversed pattern in the 1-D case, the resulting ordering is not hierarchical. In our example, all red points in the first level are ordered in the first half, but at the second level, the colors associated with the 00 reds will be in the first quarter of the ordering, while the colors associated with the 11 reds will be in the fourth quarter of the ordering. Since the hierarchical ordering is critical for reusing previous work, we order the four sublattices not in the natural order (00,01,10,11) but in a red black order: (00 11 01 10). Algorithm 1 produces this Red-Black reordering in d dimensions.

A more computationally convenient way to obtain the RB permutation is based on the fact that every point on the stencil has neighbors of opposite color. In other words, $\mathit{color}([x_1, \ldots, x_d]) = \neg \mathit{color}([x_1, \ldots, x_d] \pm e_j)$, where e_j is the unit row-vector in the j dimension, $j = 1, \ldots, d$, and $\neg$ is the logical not. With two points per dimension, in one dimension the colors are $c_1 = [0, 1]$. Inductively, if the colors in dimension $d - 1$ are c_{d-1}, the second $d - 1$ plane in dimension d will have the opposite colors, and thus: $c_d = [c_{d-1}, \neg c_{d-1}]$. Therefore, we can create the RB with only a check per point instead of counting coordinate bits. This is shown in Algorithm 2.

Algorithm 1 Red-Black order of the 2^d torus (slow)

```
$RB = \text{bitarray}(2^d, d)$
$reds = 0,\ blacks = 2^{d-1}$
for $i = 0 : 2^d - 1$
    if dec2bin($i$) has even number of bits
        $newbits = \text{dec2bin}(reds, d)$
        $reds = reds + 1$
    else
        $newbits = \text{dec2bin}(blacks, d)$
        $blacks = blacks + 1$
    $RB(i, :) = newbits$
```

Algorithm 2 Red-Black order of the 2^d torus (fast)

```
$c_0 = 0$
for $j = 1 : d$
    $c_j = [c_{j-1}, \neg c_{j-1}]$
$RB = \text{bitarray}(2^d, d)$
$reds = 0,\ blacks = 2^{d-1}$
for $i = 0 : 2^d - 1$
    if $c_d(i) == 0$
        $newbits = \text{dec2bin}(reds, d)$
        $reds = reds + 1$
    else
        $newbits = \text{dec2bin}(blacks, d)$
        $blacks = blacks + 1$
    $RB(i, :) = newbits$
```

We are now ready to combine the Red-Black reordering with the bit-reversing scheme to address the d-dimensional case. First, assume that the lattice has the same size in each dimension, i.e., $k_j = k, \forall j = 1, \ldots, d$. Then the needed permutation is given by Algorithm 3.

3.1.12 Lattices with different sizes per dimension

At every recursive level, our algorithm splits the size of each dimension in half (removing one bit), until there is only 1 node per dimension. When the dimensions do not all have the same size, some of the dimensions reach 1 node first and beyond that point they are

Algorithm 3 Hierarchical permutation of the lattice – case $k_j = k$

```
% Input:
% the coordinates of a point $x = (x_1, x_2, \ldots, x_d)$
% the global $RB$ array produced by Algorithm 2
% Output:
% The location in which $x$ is found in the hierarchical permutation
function loc = LatticeHierPermutation0($(x_1, x_2, \ldots, x_d)$)
    % Make a $d \times k$ table of all the coordinate bits
    for $j = 1 : d$
        $(b_k^j, \ldots, b_2^j, b_1^j) = \text{dec2bin}(x_j)$
    % Accumulate bit-reversed order. Start from LSB
    $loc = [\ ]$
    for $i = 1 : k$
        % A vertical section of bits. Take the i-th bit of all coordinates
        % and permute it to the corresponding red-black order
        $(s^1, \ldots, s^d) = RB(\text{bin2dec}(b_i^1, b_i^2, \ldots, b_i^d))$
        % Append this string to create the reverse order string
        $loc = [loc,\ \ (s^1, \ldots, s^d)]$
    return bin2dec($loc$)
```

not subdivided. If m out of d dimensions have reached size 1, the above algorithm should continue as in a $d-m$ dimension lattice, at every level concatenating only the active $d-m$ bits in *loc*. In this case, however, the red-black permutation RB should correspond to that of a $d-m$ dimensional lattice. The following three lemmas and the theorem, whose proofs are in the Appendix, allow us to avoid computing and storing RB_j for each $j = 1, \ldots, d$. As before, we consider c_d the array of 0/1 colors of the 2-point, d-dimensional torus.

Lemma 3.2 *For any $d > 0$, $c_d(2i) = c_{d-1}(i),\ i = 0, \ldots, 2^{d-1} - 1$.*

Lemma 3.3 *For any $d > 0$, $c_d(2i) = \neg c_d(2i+1),\ i = 0, \ldots, 2^{d-1} - 1$.*

Lemma 3.4 *For any $d > 0$ the values of $RB_d(i),\ i = 0, \ldots, 2^d - 1$ are given by:*

$$RB_d(i) = \begin{cases} \lfloor i/2 \rfloor, & \text{if } c_d(i) = 0 \\ \lfloor i/2 \rfloor + 2^{d-1}, & \text{if } c_d(i) = 1 \end{cases}.$$

We can now show how $RB_m, m < d$, can be obtained from RB_d.

Theorem 3.2 *Let RB_d be the permutation array that groups together the same colors in a red-black ordering of the two-point, d dimensional lattice, as produced by Algorithm 2. For any $0 < m < d$,*

$$RB_m(i) = \lfloor RB_d(i2^{d-m})/2^{d-m} \rfloor, \; i = 0, \ldots, 2^m - 1.$$

The theorem says that given RB_d in bit format, RB_m is obtained as the left (most significant) m bits of every 2^{d-m} number in RB_d. We now have all the pieces needed to modify Algorithm 3 to produce the permutation of the hierarchical coloring of d dimensional lattice torus of size $N = \prod_{j=1}^{d} 2^{k_j}$.

For $d > 1$, Algorithm 4 differs from the general approach discussed in the beginning of Section 3.1.8 because it pre-splits groups of colors into conformal lattices before it induces the new connectivity. The difference in the actual number of colors, however, is small. At level $i = 0, 1, \ldots$, Algorithm 4 performs a distance-$2^{i+1} - 1$ coloring and produces 2^{di+1} colors.

For classical probing, the minimum number of colors required for distance-$2^{i+1} - 1$ coloring of lattices is not known for $d > 2$ [32]. An obvious lower bound is the number of lattice sites in the "unit sphere" of graph diameter 2^i. If $\left(\begin{array}{c} d \\ i \end{array}\right)$ denotes the binomial coefficient, with a 0 value if $d < i$, the lower bound is given by [26, Theorem 2.7]:

$$\sum_{i=0}^{d} \left(\begin{array}{c} d \\ i \end{array} \right) \left(\begin{array}{c} d - i + 2^{i-1} \\ d \end{array} \right).$$

For sufficiently large distances, this is $O(2^{3i-1}/3)$ for $d = 3$, and $O(2^{4i-3}/3)$ for $d = 4$. Thus, we can bound asymptotically how many more colors our method gives:

$$\frac{\text{Number of colors in hierarchical probing}}{\text{Number of colors in classical probing}} \begin{cases} < 12, & \text{if d=3} \\ < 48, & \text{if d=4.} \end{cases}$$

Algorithm 4 Hierarchical permutation of the lattice – case $2^{k_i} \neq 2^{k_j}$

```
% Input:
% the coordinates of a point $x = (x_1, x_2, \dots, x_d)$
% the global $RB$ array produced by Algorithm 2
% Output:
% The location in which $x$ is found in the hierarchical permutation
function loc = LatticeHierPermutation($(x_1, x_2, \dots, x_d)$)
    % Make a $d \times \max(k_j)$ table of all the coordinate bits
    % Dimensions with smaller sizes only have up to $k_j$ bits set
    for $j = 1 : d$
        $(b^j_{k_j}, \dots, b^j_2, b^j_1) = \text{dec2bin}(x_j)$
    % Accumulate bit-reversed order. Start from LSB
    $loc = [\ ]$
    for $i = 1 : \max(k_j)$
        % A vertical section of bits. Take the i-th bit of all coordinates
        % in dimensions that can still be subdivided $(i \leq k_j)$.
        % Record number of such dimensions
        $activeDims = 0$
        $bits = [\ ]$
        for $j = 1 : d$
            if $(i \leq k_j)$
                $bits = [bits, \quad b^j_i]$
                $activeDims = activeDims + 1$
        % permute it to the corresponding red-black order using $RB_i$
        $index = \text{bin2dec}(bits) 2^{d - activeDims}$
        $(s^1, \dots, s^{activeDims}) = \lfloor RB(index)/2^{d-activeDims} \rfloor$
        % Append this string to create the reverse order string
        $loc = [loc, \quad (s^1, \dots, s^{activeDims})]$
    return bin2dec($loc$)
```

In practice, we have observed ratios of 2–3. On the other hand, because hierarchical probing uses more vectors, the variance reduction it achieves when a certain distance coloring completes, i.e., after 2^{di+1} quadratures in the MC estimator, is typically better than classical probing for the same distance.

In terms of computational cost, this algorithm is very efficient, especially when compared with classical probing. As an example, producing the hierarchical permutation of a 64^4 lattice takes about 6 seconds on a Macbook Pro with 2.8 GHz Intel Core 2 Duo. More importantly, the permutation of each coordinate is obtained independently which facilitates parallel computing.

3.1.13 Coloring lattices with non-power of two sizes

Consider a lattice of size $N = \prod_{j=1}^{d} n_j$. Sometimes, LQCD may generate lattices where one or more n_j are not powers of two. In this case, it is typical that $n_j = 2^m p$, where $p \neq 2$ is a small prime number. Our hierarchical coloring method works up to m levels, but then the remaining subgrids are of odd size in the j-th dimension, causing coloring conflicts because of wrap-around. In the next theorem, whose proof is in Appendix, we show that such a lattice is three colorable.

Theorem 3.3 *A toroidal, uniform lattice of size $N = \prod_{j=1}^{d} n_j$, where one or more n_j are odd, admits a three-coloring with point $x = (x_1, \ldots, x_d)$ receiving color:*

$$C(x) = \left(\sum_{j=1}^{d} x_j + \sum_{j=1}^{d} \delta(x_j) \right) \mathit{mod}\, 3, \ \ \mathit{where}\ \delta(x_j) = \begin{cases} 1, & \mathit{if}\ (x_j = n_j - 1)\ \mathit{and} \\ & \quad (n_j - 1\ \mathit{mod}\, 3 = 0) \\ 0, & \mathit{everywhere\ else.} \end{cases}$$

After the three-coloring is produced, further hierarchical colorings are not practical, since the coloring yields blocks of nodes that are not conformal lattices, and are of irregular shapes. This prevents the use of a method similar to the one described in section 3. Because of this, for lattices which have dimensions with factors other than two, we can proceed only one level further after the factors of two have been exhausted by the hierarchical coloring algorithm. This is not a shortcoming in LQCD, since, by construction, lattices have dimensions with at most one odd factor. Finally, note that the number of hierarchical probing vectors produced before exhausting the powers of two in each dimension is typically large, obviating the need for a last three-coloring step.

3.1.14 Generating the probing basis

Assume for the moment each color group at any level of the hierarchical method is colored with exactly two colors. Consider also the permuted matrix $A(perm, perm)$ so that colors appear in the block diagonal. In Section 3.1.7, we saw the Hadamard vectors required for probing the first two levels of this recursion for a 8×8 matrix: $[h_0, h_4]$ and $[h_0, h_4, h_2, h_6,]$. If $\otimes$ denotes the Kronecker product and $\mathbf{1}_k = [1, \ldots, 1]^T$ the vector of k ones, these can be written as:

$$\begin{aligned}
[h_0, h_4] &= H_2 \otimes \mathbf{1}_4, \\
[h_0, h_4, h_2, h_6] &= [H_2 \otimes H_2(:,1),\ H_2 \otimes H_2(:,2)] \otimes \mathbf{1}_2.
\end{aligned}$$

This pattern extends to any recursion level $i = 1, 2, \ldots, \log_2 N$. If we denote by $Z^{(i)}$ the Hadamard vectors that span the i-th level probing vectors, these are obtained by the following recursion:

$$\begin{aligned}
\tilde{Z}^{(1)} &= H_2, \\
\tilde{Z}^{(i)} &= \left[\tilde{Z}^{(i-1)} \otimes H_2(:,1),\ \tilde{Z}^{(i-1)} \otimes H_2(:,2)\right], \\
Z^{(i)} &= \tilde{Z}^{(i)} \otimes \mathbf{1}_{N/2^i}. \qquad (3.5)
\end{aligned}$$

Intuitively, this says that at every level, we should repeat the pattern devised in the previous level to double the domains for the first 2^{i-1} vectors (Kronecker product with $[1, 1]^T$), and then should split each basic subdomain in two opposites (Kronecker product with $[1, -1]^T$).

The hierarchy $Z^{(i-1)} = Z^{(i)}(:, 1 : 2^{i-1})$ implies that quadratures performed with $Z^{(i-1)}$ can be reused if we need to increase the probing level. To obtain the m-th probing vector, therefore, we can consider the m-th vector of $Z^{(\log_2 N)}$. Its rows can be constructed piece by piece recursively through (3.5) and without constructing all $Z^{(\log_2 N)}$. In fact, we can even avoid the recursive construction and compute any arbitrary element of $Z^{(\log_2 N)}(j, m)$ directly. This is useful in parallel computing where each processor generates only the local rows of this vector. The reason is that recursion (3.5) produces a known permutation of the natural order of the Hadamard matrix, specifically the column indices are:

$$0, N/2, N/4, 3N/4, N/8, 5N/8, 3N/8, 7N/8, \ldots. \tag{3.6}$$

We can compute a-priori this column permutation array, $Hperm$, for all N, or for as many vectors as we plan to use in the MC estimator. Then by using the inverse permutation ($iperm$) associated with the given hierarchical coloring, the j-th element of the m-th probing vector can be computed directly through (3.2) as:

$$z_m(j) = H_N(iperm(j), Hperm(m)). \tag{3.7}$$

We observe now that the assumption that each subgroup is colored with exactly two colors is not necessary. The ordering given in (3.6) is the same if each subgroup is colored by any power of two colors, which could be different at different levels. The sequence (3.6) is built on the smallest increment of powers of two and thus subsumes any higher powers.

We can extend the above ideas to generate the probing basis for arbitrary N, when at every level each color block is split into exactly the same (possibly non-power of two) colors. For example, at the first level we split the graph into 3 colors, at level two, each of the 3 color blocks is colored with exactly 5 colors, at level three, each of the 5 color blocks is colored with exactly 2 colors, and so on. The problem is that Hadamard matrices do not exist for arbitrary dimensions. For example, for 3 probing vectors, there is no orthogonal basis Z of ± 1 elements, such that $ZZ^T = I$. In this general case, we must resort to the N-th roots of unity, i.e., the Fourier matrices F_n.

Assume that the number of colors at level i is $c(i)$ for all blocks at that level, then the probing basis is constructed recursively as:

$$\begin{aligned}
\tilde{Z}^{(1)} &= F_{c(1)}, \\
\tilde{Z}^{(i)} &= \left[\tilde{Z}^{(i-1)} \otimes F_{c(i)}(:,1), \ldots, \tilde{Z}^{(i-1)} \otimes F_{c(i)}(:,c(i))\right], \\
Z^{(i)} &= \tilde{Z}^{(i)} \otimes \mathbf{1}_{N/\gamma_i}, \quad \text{where } \gamma_i = \prod_{j=1}^{i} c(j).
\end{aligned} \tag{3.8}$$

By construction, the vectors of $Z^{(i-1)}$ are contained in $Z^{(i)}$, and any arbitrary vector can be generated with a simple recursive algorithm. However, we have introduced complex arithmetic which doubles the computational cost for real matrices. On the other hand, if a $c(i)$ is a power of two, its $F_{c(i)}$ can be replaced by $H_{c(i)}$. This can be useful when the non-power of two colors appear only at later recursion levels for which the number of probing vectors is large and may not be used, or when only one or two $F_{c(i)}$ will suffice.

To summarize, we have provided an inexpensive way to generate, for any matrix size, an arbitrary vector of the hierarchical probing sequence through (3.8), as long as the number of colors is the same within the same level for each subgraph. If, in addition, the matrix size and the color numbers are powers of two, (3.6–3.7) provide an even simpler way to generate the probing sequence. In LQCD, many of the lattices fall in this last category.

3.1.15 Removing the deterministic bias

The probing vectors produced in Section 3.1.14 are deterministic and, even though they give better approximations than random vectors, they introduce a bias. To avoid this, we can view formula (3.5) not as a sequence of vectors but *as a process of generating an orthogonal basis starting from any vector and following a particular pattern.* Therefore, consider a random vector $z_0 \in \mathbb{Z}_2^N$, and $[z_1, z_2, \ldots, z_m]$ the sequence of vectors produced by (3.5). If $\odot$ is the element-wise Hadamard product, the vectors built as

$$V = [z_0 \odot z_1, z_0 \odot z_2, \ldots, z_0 \odot z_m] \tag{3.9}$$

have the same properties as Z, i.e., $V^T V = Z^T Z$ and VV^T has same non-zero pattern as ZZ^T ($VV^T = (z_0 z_0^T) \odot ZZ^T$), but one can easily show that the expected value of the trace estimate over all z_0 is the matrix trace, yielding an unbiased estimator.

3.1.16 Numerical experiments

We present a set of numerical examples on control test problems and on a large QCD calculation in order to show the effectiveness of hierarchical probing over classical probing, and over standard noise Monte Carlo estimators for $\mathbf{Tr}(A^{-1})$. We also study the effect of the unbiased estimator.

Our standard control problem is the discretization of the Laplacian on a uniform lattice with periodic boundary conditions. We control the dimensions (3-D or 4-D), the size per dimension, and the conditioning (and thus the decay of the elements of the inverse) by applying a shift to the matrix. Most importantly, for these matrices we know the trace of the inverse analytically. We will refer to such problems as Laplacian, with their size implying their dimensionality.

3.1.17 Comparison with classical probing

For this set of experiments we consider a 64^3 Laplacian, shifted so that its condition number is on the order of 10^2. Therefore, its A^{-1} exhibits dominant features on and close to (in a graph theoretical sense) the non-zero structure of A, with decay away from it. The decay rate depends on the conditioning of A. Our methods should be able to pick this structure effectively.

Figure 3.4 shows the performance of classical probing, which is a natural benchmark for our methods. The left graph shows that for larger distance colorings, probing performs extremely well. For example, with 317 probing vectors, which correspond to a 8-distance coloring, we achieve more than two orders reduction in the error. Of course, if the approximation is not good enough, this work must be discarded, and the algorithm must be repeated for higher distances. Hadamard vectors, in their natural order, do not capture well the nonzero structure of this A.

The right graph in Figure 3.4 shows one way to improve accuracy beyond a certain probing distance. After using $[0, \ldots, 0, \mathbf{1}_{r(m)}^T, 0, \ldots 0]^T$ as the probing vector for color m, we continue building a Hadamard matrix in its natural order only for the $r(m)$ coordinates of that color. If probing has captured the most important parts of the matrix, the remaining parts could be sufficiently approximated by natural order Hadamard vectors. This is confirmed by the results in the graph, if one knows what initial probing distance to pick. On the other hand, hierarchical probing, which considers all possible levels, achieves better performance than all other combinations.

In Figure 3.5, left graph, we stop our recursive algorithm at various levels and use the resulting permutation to generate the vectors for the trace computation. It is clearly beneficial to allow the recursion to run for all levels. We also point out that stopping at intermediate levels behaves similarly to classical probing with the corresponding distance. On the right graph of Figure 3.5, we observe no difference between methods for high conditioned matrices. The reason is that the eigenvector of the smallest eigenvalue of A is the vector of all ones, $\mathbf{1}_N$. The more ill conditioned A is, the more A^{-1} is dominated by $\mathbf{1}_N \mathbf{1}_N^T$, which has absolutely no variation or pattern.

We point out that the experiments in this subsection did not use the unbiased estimator of Section 3.1.15. This has a severe effect for the Laplacian matrix because the first vector of our Hadamard sequences is $h_0 = \mathbf{1}_N$, the lowest eigenvector. Even for a well conditioned Laplacian, starting with h_0 guarantees that the first trace estimate will have no contribution from other eigenvectors, and thus will have a large error. From a statistical

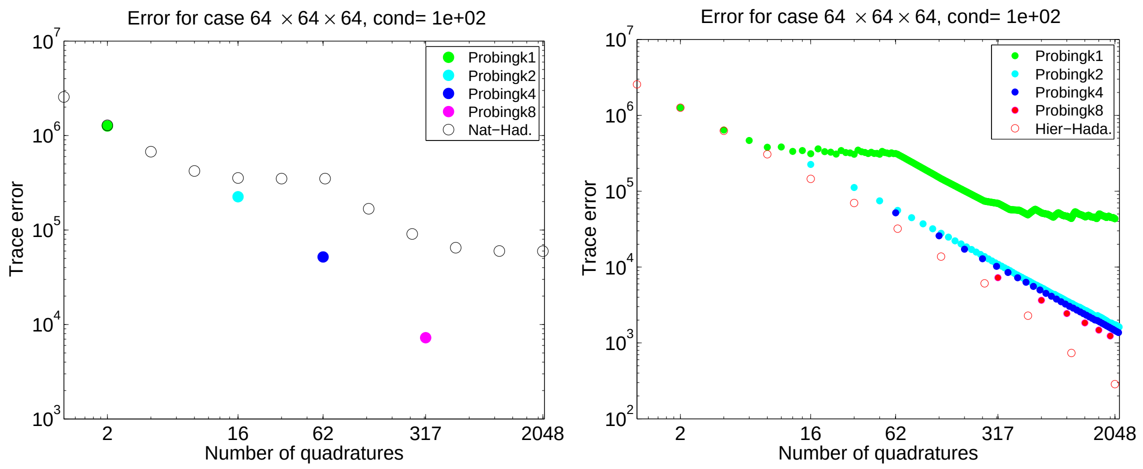


Figure 3.4: Error in the $\mathbf{Tr}(A^{-1})$ approximation using the MC method with various deterministic vectors. Classic probing requires 2,16,62, and 317 colors for probing distances 1,2,4, and 8, respectively. Left: Classic probing approximates the trace better than the same number of Hadamard vectors taken in their natural order. Going to higher distance-k requires discarding previous work. Right: Perform distance-k probing, then apply Hadamard in natural order within each color. Performs well, but hierarchical performs even better.

point of view, h_0 is the worst starting vector for Laplacians, but it better exposes the rate at which various methods reduce error.

3.1.18 Comparison with random-noise Monte Carlo

Having established that hierarchical probing discovers matrix structure as well as classical probing, we turn to gauge its improvements over the standard $\mathbb{Z}_2$ noise MC estimator. First, we show three sets of graphs for increasing condition numbers of the Laplacian. We use the 64^3, 32^4, and 64×128^2 lattices, and plot the convergence of the trace estimates for hierarchical probing, natural order Hadamard, and for the standard $\mathbb{Z}_2$ random estimator. Both Hadamard sequences employ the bias removing technique (3.9). As it is typical, the random estimator includes error bars designating the two standard deviation confidence intervals, $\pm 2(\overline{Var}/s)^{1/2}$, where $\overline{Var}$ is the variance estimator.

Figure 3.6 shows the convergence history of the three estimators for well conditioned shifted Laplacians, which therefore have prominent structure in A^{-1}. Hierarchical probing

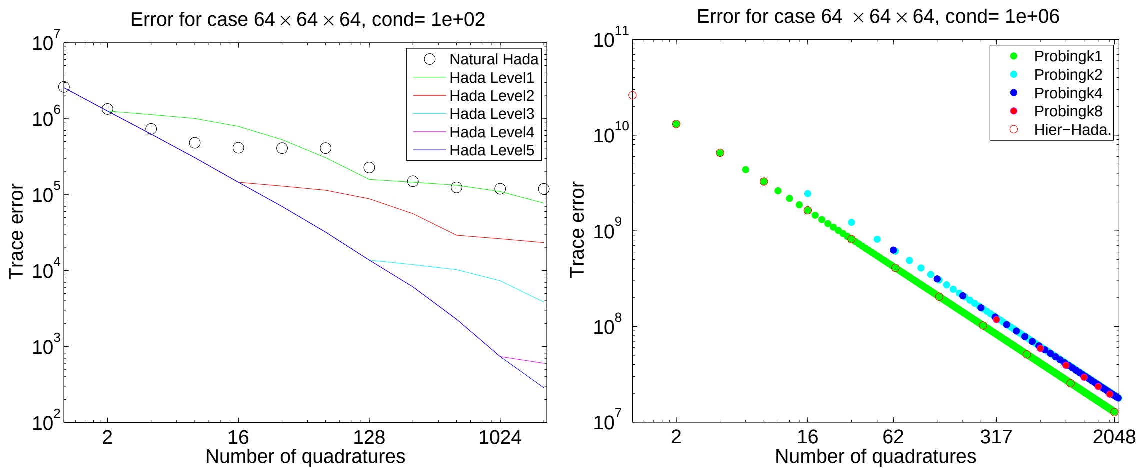


Figure 3.5: Left: The hierarchical coloring algorithm is stopped after $1, 2, 3, 4, 5$ levels corresponding to distances $2, 4, 8, 16, 32$. The ticks on the x-axis show the number of colors for each distance. Trace estimation is effective up to the stopped level; beyond that the vectors do not capture the remaining areas of large elements in A^{-1}. Compare the results with classical probing in Figure 3.4, which requires only a few less colors for the same distance. Right: When the matrix is shifted to have high condition number, the lack of structure in A^{-1} causes all methods to produce similar results.

exploits this structure, and thus performs much better than the other methods. Note that the problem on the left graph is identical to the one used in the previous section. The far better performance of the Hadamard sequences in this case is due to avoiding the eigenvector h_0 as the starting vector.

Figures 3.7 and 3.8 show results as the condition number of the problems increase. As expected, the advantage of hierarchical probing wanes as the structure of A^{-1} disappears, but there is still no reason not to use it as diminishing improvement remain evident. We have included 4-D lattices in our experiments, first because of their use in LQCD, and second because their structure is more difficult to exploit than lower dimensional lattices. For 1-D or 2-D lattices which we do not show, hierarchical probing was significantly more efficient.

Once we use a random vector z_0 to modify our sequence as in (3.9), hierarchical probing becomes a stochastic process, whose statistical properties must be studied. Thus, we generate $z_0^{(i)}, i = 1 : 100$, $\mathbb{Z}_2$ random vectors, and for each one we produce a modified

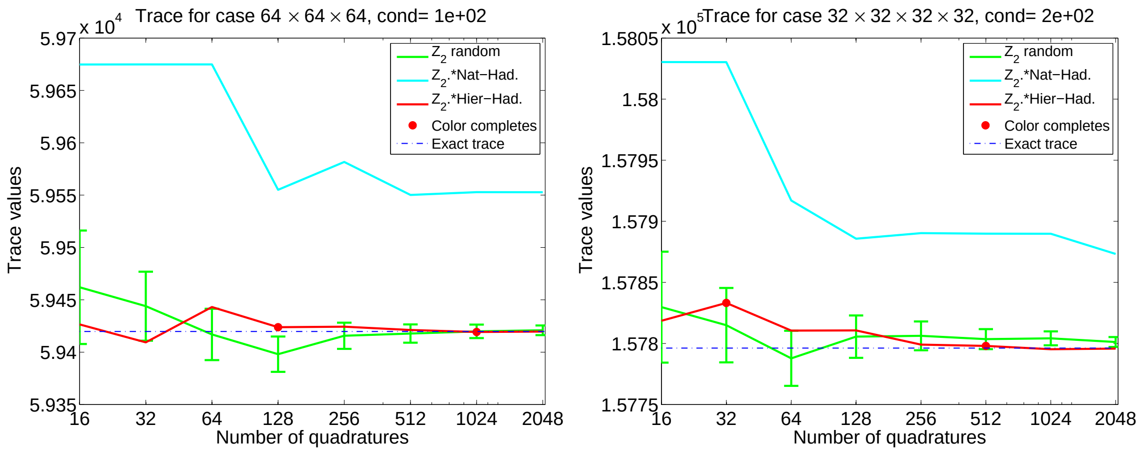


Figure 3.6: Convergence history of $\mathbb{Z}_2$ random estimator, Hadamard vectors in natural order, and hierarchical probing, the latter two with bias removed as in (3.9). Because of small condition number, A^{-1} has a lot of structure, making hierarchical probing clearly superior to the standard estimator. As expected, Hadamard vectors in natural order are not competitive. The markers on the plot of the hierarchical probing method designate the number of vectors required for a particular distance coloring to complete. It is on these markers that structure is captured and error minimized.

sequence of the hierarchical probing vectors. Then, we use the 100 values $x_m^T A^{-1} x_m$, where $x_m = z_0^{(i)} \odot z_m$, at every step of the 100 MC estimators to calculate confidence intervals. These are shown in Figure 3.9. We emphasize that the confidence intervals for the $\mathbb{Z}_2$ random estimator are computed differently, based on the $\overline{Var}$ estimator of the preceding MC steps, so they may not be accurate initially. Even on a 4-D problem, hierarchical probing provides a clear variance improvement.

3.1.19 A large QCD problem

The methodology presented in this paper has the potential of improving a multitude of LQCD calculations. In this section, we focus on the calculation of $C = \textbf{Tr}(D^{-1})$, where the Dirac matrix D is a non-symmetric complex sparse matrix. This is representative of a larger class of calculations usually called "*disconnected diagrams*". The physical observable C is related to an important property of QCD called spontaneous chiral symmetry breaking.

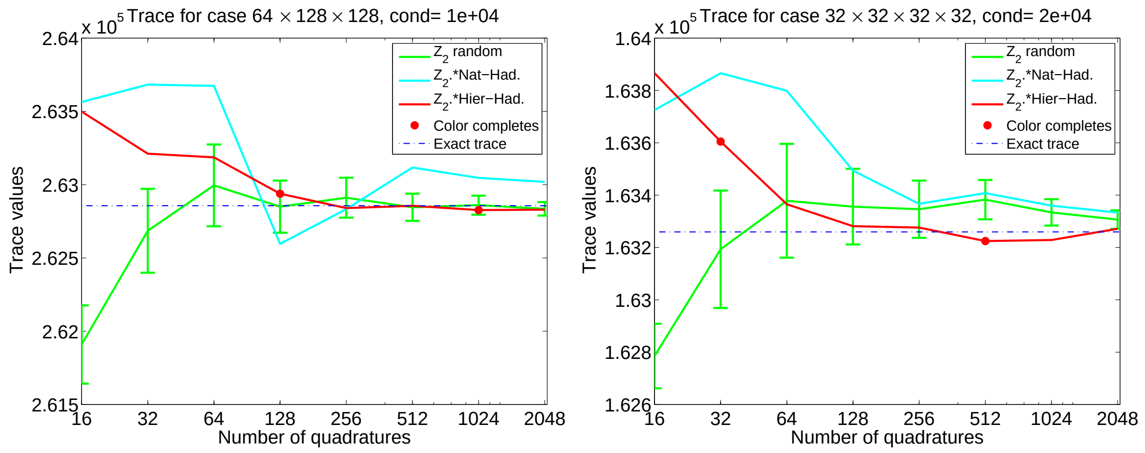


Figure 3.7: Convergence history of the three estimators as in Figure 3.6 for a larger condition number $O(10^4)$. As the structure of A^{-1} becomes less prominent, the differences between methods reduce. Still, hierarchical probing has a clear advantage.

Our goal is to compare the standard MC approach of computing the trace with our hierarchical probing method. Our test was performed on a single gauge field configuration using the Dirac matrix that corresponds to the "*strange*" quark Dirac matrix resulting from the Clover-Wilson fermion discretization [56]. The *strange* quark is the third heaviest quark flavor in nature. The gauge configuration had dimensions of $32^3 \times 64$ with a lattice spacing of $a = 0.11fm$, for a problem size of 24 million.

First, we used an ensemble of $n = 253$ noise vectors to estimate the variance of the standard MC method, with complete probing (dilution) of the internal color-spin space of dimension 12 to completely eliminate the variance due to connections in this space. Then, for each of these noise vectors, we modified as in (3.9) a sequence of hierarchical probing vectors which were generated based on space-time connections. As with the standard MC estimator, full dilution of the color-spin space was performed. This procedure was performed in order to statistically estimate the variance of hierarchical probing, similarly to the test in Figure 3.9. In Figure 3.10(a), we present the variance of the hierarchical probing estimator as a function of the number of space-time probing vectors in the sequence. The main feature in this plot is that the variance drops as more vectors are used. Local minima occur at numbers of vectors that are powers of 2, where all connections of a given

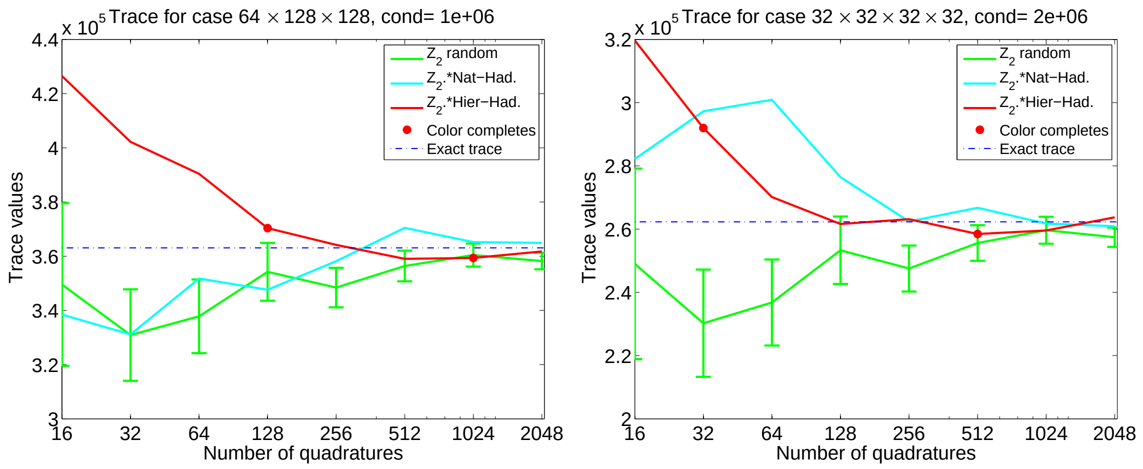


Figure 3.8: Convergence history of the three estimators as in Figure 3.6 for a high condition number $O(10^6)$. Even with no prominent structure in A^{-1} to discover, hierarchical probing is as effective as the standard method.

Manhattan distance are eliminated from the variance. The uncertainty of the variance, represented by the errorbars in the plot, is estimated using the Jackknife resampling procedure of our noise vector ensemble.

In addition to the variance, we estimate the speed-up ratio of the hierarchical probing estimator over the standard MC estimator. We define speed-up ratio as:

$$R_s = \frac{V_{stoc}}{V_{hp}(s)\ \times\ s}\,,$$

where $V_{hp}(s)$ is the variance over the n different runs when the s-th hierarchical probing vector is used, and V_{stoch} is the variance of the standard MC estimator as estimated from $n = 253$ samples. The rescaling factor of s is there to account for the fact that if one had been using a pure stochastic noise with $n \times s$ vectors, the variance would be smaller by a factor of s. Thus, the variance comparison is performed on equal amount of computation for both methods. In Figure 3.10(b) we present the speed-up ratio R_s as a function s. The errorbars on R_s are estimated using Jackknife resampling from our ensemble of starting noise vectors. The peaks in this plot occur at the points where s is a power of 2, as in the variance case. A maximum overall speed-up factor of about 10 is observed at $s = 512$.

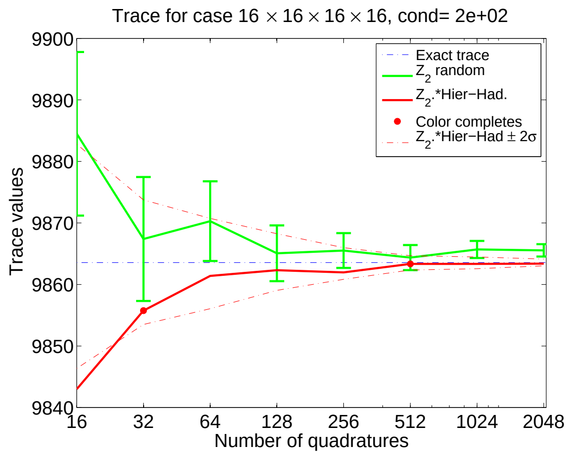


Figure 3.9: Providing statistics over 100 random vectors z_0, used to modify the sequence of 2048 hierarchical probing vectors as in (3.9). At every step, the variance of quadratures from the 100 different runs is computed, and confidence intervals reported around the hierarchical probing convergence. Note that for the standard noise MC estimator confidence intervals are computed differently and thus they are not directly comparable.

Note that the color completion points for this experiment are at $s = 2$, $s = 32$ and $s = 512$ vectors.

Finally, we report on a comparison with classical probing for this large QCD problem. There is a variety of approaches for efficient distance-2 coloring in the literature [38, 33], but we have not found any standard approaches for distance-k coloring. On lattices, however, the distance-k neighborhood of a node is explicitly known geometrically. We implemented a coloring algorithm that visits only this neighborhood for each node, thus achieving the minimum possible complexity for this problem [33]. Specifically, for each node, we make a list of colors previously assigned to its distance-k neighbors, and pick the smallest color number not appearing in the list. The distance-4 coloring of our LQCD lattice produced 123 colors and took 457 seconds on an Intel Xeon X5672, 3.2GHz server. Using four random vectors with the structure of each of these colors (so that the total number of quadratures is similar to our hierarchical probing), we ran 50 sets of experiments, and measured the variance of classical probing. We found that its variance was 2.16 times larger than our hierarchical probing, or in other words, our method was 2.16

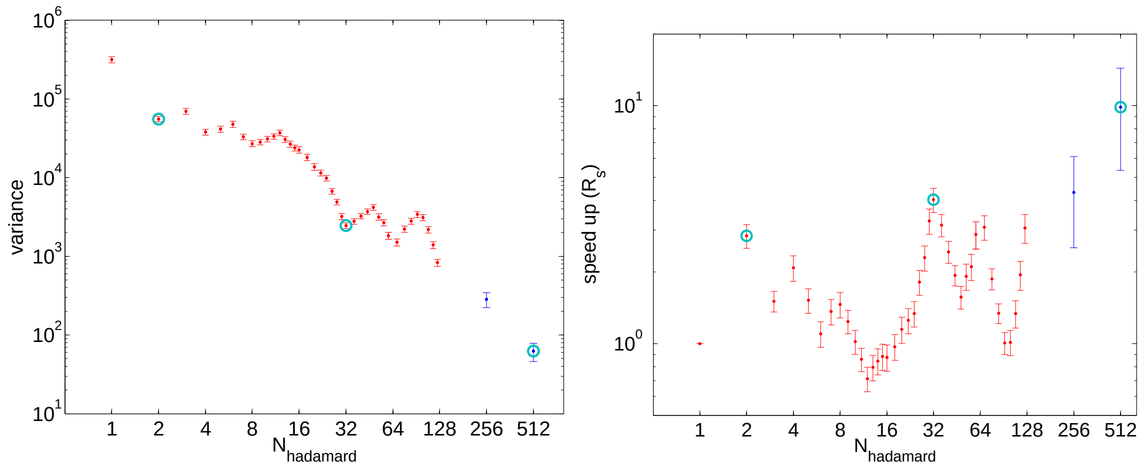


Figure 3.10: (a) Left: The variance of the hierarchical probing trace estimator as a function of the number of vectors (s) used. The minima appear when s is a power of two. The places where the colors complete are marked with the cyan circle. These minima become successively deeper as we progress from 2 to 32 to 512 vectors. (b) Right: Speed-up of the LQCD trace calculation over the standard $\mathbb{Z}_2$ MC estimator. The cyan circles mark where colors complete. The maximal speed up is observed at $s = 512$. In both cases the uncertainties are estimated using the Jackknife procedure on a sample of 253 noise vectors, except for $s = 256$ and 512 where 37 vectors were used.

times faster. This is expected as we explained earlier. Finally, note that computing the quadratures took 4 hours on four GPUs, on a dedicated machine for LQCD calculations. Even though classic probing with distance-4 is feasible for this problem, computing the distance-8 coloring requires 5377 seconds, which becomes comparable to the time for computing the quadratures. Contrast that to the 2 seconds needed to compute the hierarchical probing.

3.1.20 Conclusions

The motivation for this work comes from our need to compute $\mathbf{Tr}(A^{-1})$ for very large sparse matrices and LQCD lattices. Current methods are based on Monte Carlo and do not sufficiently exploit the structure of the matrix. Probing is an attractive technique but cannot be used incrementally, and becomes expensive for ill conditioned problems. Our research has addressed these issues.

We have introduced the idea of hierarchical probing that produces suboptimal but nested distance-2^i colorings recursively, for all distances up to the diameter of the graph. We have adapted this idea to uniform lattices of any dimension in a very efficient and parallelizable way.

To generate probing vectors that follow the hierarchical permutation and can be used incrementally to improve accuracy, we have developed an algorithm that produces a specific permutation of the Hadamard vectors. This algorithm is limited to cases where the number of colors produced at every level is a power of two. We have also provided a recursive algorithm based on Fourier matrices that provides the appropriate sequence under the weaker assumption of having the same number of colors per block within a single level. These conditions are satisfied on toroidal lattices. Finally, we proposed an inexpensive technique to avoid deterministic bias while using the above sequences of vectors.

We have performed a set of experiments in the context of computing $\mathbf{Tr}(A^{-1})$, and have shown that providing a hierarchical coloring for all possible distances is to be preferred over classical probing for a specific distance. We also showed that our methods provide significant speed-ups over the standard Monte Carlo approach.

Currently we are working to extend the idea of hierarchical coloring to general sparse matrices, and to combine it with other variance reduction techniques, in particular deflation type methods.

APPENDIX

Lemma 3.2. **Proof:** We use induction on d. For $d = 2$, $c_2 = [0, 1, 1, 0]$ and the result holds. Assume the result holds for any dimension $d-1$ or lower. Then for d dimensions, since the first half of c_d is the same as c_{d-1}, for $i = 0, \dots 2^{d-1} - 1$, we have

$$\begin{aligned} c_d(2i) &= \neg c_d(2i - 2^{d-1}) = \neg c_{d-1}(2i - 2^{d-1}) && \text{(recursive definition of } c_d\text{)} \\ &= \neg c_{d-2}(i - 2^{d-2}) = \neg c_d(i - 2^{d-2}) && \text{(inductive hypothesis)} \\ &= \neg c_{d-2}(i - 2^{d-2}) = \neg(\neg c_{d-1}(i)) = c_{d-1}(i) && \text{(recursive definition of } c_d\text{).} \end{aligned}$$

■

Lemma 3.8. **Proof:** Because c_d are the colors of the two-point, d dimensional torus, every even point $2i$ is the beginning of a new 1-D line and thus has a different color from its neighbor $2i + 1$. It can also be proved inductively, since by construction $2i$ and $2i + 1$ cannot be split across c_{d-1} and c_d. ■

Lemma 3.4. **Proof:** Because of Lemma 3.8, after every pair of indices $(2i, 2i + 1)$ is considered, the number of reds or blacks increases only by 1. Algorithm 3 sends all red ($c_d(i) = 0$) points i to the first half of the permutation in the order they are considered, which increases by 1 every two steps. Hence the first part of the equation. Black colors are sent to the second half, which completes the proof. ■

Theorem 3.2. **Proof:** We show first for $m = d - 1$. Because of Lemma 3.2, we consider the even points in RB_d. Assume first $c_d(2i) = c_{d-1}(i) = 0$. From Lemma 3.4 we have, $RB_d(2i) = \lfloor 2i/2 \rfloor = i$. Then, $RB_{d-1}(i) = \lfloor i/2 \rfloor = \lfloor RB_d(2i)/2 \rfloor$. Now assume $c_d(2i) = c_{d-1}(i) = 1$. From Lemma 3.4 we have, $RB_d(2i) = 2^{d-1} + \lfloor 2i/2 \rfloor = 2^{d-1} + i$, and therefore $RB_{d-1}(i) = 2^{d-2} + \lfloor i/2 \rfloor = 2^{d-2} + \lfloor (RB_d(2i) - 2^{d-1})/2 \rfloor = \lfloor RB_d(2i)/2 \rfloor$, which proves the formula for both colors. A simple inductive argument proves the result for any $m = 1, \ldots, d - 2$. ■

Theorem 3.3. **Proof:** We show that $C(x) \neq C(x')$ for any two points, x, x' with $||x - x'||_1 = 1$. These two points differ by one coordinate, j, since otherwise they are no longer unit length apart. So, $C(x) - C(x') = \left(\sum_{i=1}^{N}(x_i - x'_i) + \sum_{i=1}^{N}(\delta(x_i) - \delta(x'_i))\right) mod\ 3 = \left(x_j - x'_j + \delta(x_j) - \delta(x'_j)\right)\ mod\ 3$. We consider the following cases.

If neither x and x' lie on the boundary of the j-th dimension, $x_j \neq n_j - 1$, then $\delta(x_j) = \delta(x'_j) = 0$, and $C(x) - C(x') = (x_j - x'_j)\ mod\ 3 = \pm 1\ mod\ 3 \neq 0$.

Since x_j, x'_j both vary along the j-th dimension, only one of these points can lie on the boundary point of that dimension, consequently, only one of the two delta can be equal to one. Without loss of generality, we assume that x_j is on the boundary of the j-th dimension, so $C(x) - C(x') = (x_j - x'_j + \delta(x_j))\ mod\ 3$. In this case $x_j - x'_j = 1$, or in the warp around case, where $x'_j = 0, x_j - x'_j = n_j - 1$. There are two subcases:

1. $\delta(x_j) = 0$, then $x_j = n_j - 1$ with $n_j - 1 \; mod \; 3 \neq 0$, so $C(x) - C(x') = 1$, or $C(x) - C(x') = (n_j - 1 \; mod \; 3)$ and thus is non-zero.

2. $\delta(x_j) = 1$, then $x_j = n_j - 1$ and $n_j - 1 \; mod \; 3 = 0$, so $C(x) - C(x')$ is equal to $(1 + \delta(x_j)) \; mod \; 3 = (1 + 1) \; mod \; 3 \neq 0$, or $C(x) - C(x')$ is equal to $(n_j - 1 + \delta(x_j)) = (0 + \delta(x_j)) \; mod \; 3 = 1 \; mod \; 3 \neq 0$.

■

3.2 Lattices of arbitrary dimensions

3.2.1 Introduction and Preliminaries

In the previous section, we introduced an algorithm which produced hierarchical probing vectors, for lattices with dimension lengths that are powers of two. The main idea behind the algorithm was to find a way to split a lattice quickly into a collection of sublattices recursively, as long as the length of the lattice was a 4power of two. This was achieved by splitting each lattice into 2^d sublattices, where d is the dimensionality of the lattice. Since points which share a sublattice are all distance 2 away from each other, if they are assigned the same color, a valid distance 1 coloring of that sublattice will result. Further, if the lattice dimensions are a power of two, the lattice will split evenly, so each color will be divided into the same number of colors, and thus will be hierarchical. This process can then be continued recursively, until the desired number of colors is reached, or until the every node has a unique coloring. Unfortunately, if the lattices dimensions are not powers of two, then the sublattice cannot be split evenly into 2^d sublattices. At each level the dimensions of the sublattice being split will be reduced by a factor of two. At the level at which the dimensions of the sublattice are no longer divisible by two, the algorithm will be unable to continue, since it cannot then split further into even sublattices.

In this section we extend the hierarchical probing algorithm and the associated theory to sublattices of arbitrary dimensions, as long as those dimensions share some common factors. Our algorithm bases the number of splits of the original lattice on the prime factors of the lengths of its dimensions. By using these factors the lattice can be divided up into sublattices of equal sizes, allowing the process to be continued recursively. When the factors of the lattice dimensions contain factors of 2, the original method using fast binary arithmetic can still be used. It should be noted that the number of colors produced at each hierarchical level m by our method is larger than the minimum number needed to produce a distance-m coloring of the lattice. In most cases this is an acceptable trade off in order to ensure that the hierarchical property of the generated colorings is maintained. Moreover, these additional vectors reduce the error further than the optimal coloring for the same distance.

3.3 Lattices as spans of sublattices

Formally, a lattice is a discrete additive subgroup of $\mathbb{R}^n$. Intuitively, it is a collection of points, such that adding the location of any points together, gives the coordinates of another valid point, and there is a minimum distance between the closest two points. A good example of a d-dimensional lattice would be the Cartesian product of the integers, which is the canonical d-dimensional regular grid. One can contrast this with a vector space such as the d-dimensional Cartesian product of the reals. A lattice need not be infinite, it can be formed on any finite group that has the required properties. In this paper we are mainly interested in finite lattices that have a periodic boundary condition.

Similarly to vector spaces, one can write down the basis for a lattice. We then say the lattice is generated by $\mathbf{B}$, a set of d vectors of $\mathbb{R}^d$, as

$$\mathcal{L}(\mathbf{B}) = \left\{ \sum_{i=1}^{d} x_i * \mathbf{b}_i, \quad \mathbf{b}_i \in \mathbf{B}, x_i \in \mathbb{Z} \right\} = \left\{ \mathbf{B}\mathbf{x}, \quad \mathbf{x} \in \mathbb{Z}^d \right\}. \tag{3.10}$$

Note how the requirement that the coefficients of the basis vectors be integer enforces a minimum distance between two points, in contrast to a vector space. Returning to the example of the regular grid, we write the generating function for this lattice as

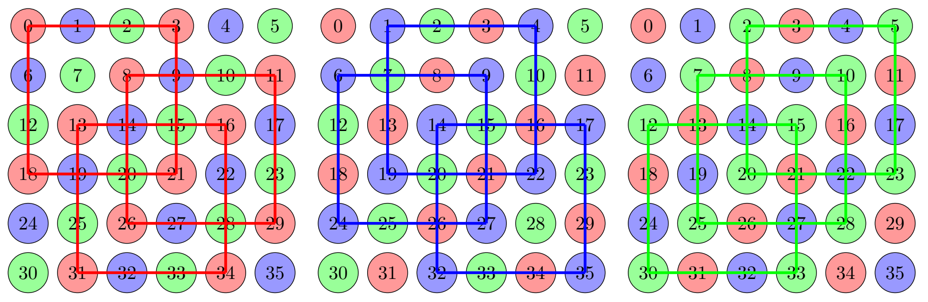
$$\mathcal{L}(\mathbf{I}) = \left\{ \sum_{i=1}^{d} x_i * e_i, \quad e_i \in \mathbf{I}, x_i \in \mathbb{Z} \right\} = \left\{ \mathbf{I}\mathbf{x}, \quad \mathbf{x} \in \mathbb{Z}^d \right\}, \tag{3.11}$$

where $\mathbf{I}$ is the d-dimensional identity matrix and e_i is the i-th column of $\mathbf{I}$.

Just as vector spaces may contain closed sub-spaces, lattices may contain closed sub-lattices. In this paper we are interested in the sublattices of $\mathcal{L}(\mathbf{I})$. In particular, we want to determine which sublattices of $\mathcal{L}(\mathbf{I})$ a point lies in. For example, the sublattice $\mathcal{L}(b\mathbf{I})$ can describe only $\frac{1}{b}$ of the points of $\mathcal{L}(\mathbf{I})$, with spacing b. Consider also the concept of an affine lattice, which we define as

$$\mathcal{L}(\mathbf{B})_{\mathbf{c}} = \left\{ \sum_{i=1}^{d} x_i * \mathbf{b}_i + \mathbf{c}, \quad \mathbf{b}_i \in \mathbf{B}, x_i \in \mathbb{Z}, \mathbf{c} \in \mathbb{Z}^d \right\} = \left\{ \mathbf{B}\mathbf{x}+\mathbf{c}, \quad \mathbf{x}, \mathbf{c} \in \mathbb{Z}^d \right\}. \tag{3.12}$$

We can use these to decompose $\mathcal{L}(\mathbf{I})$ into a union of affine sublattices, since non-affine sublattices represent only sublattices centered at zero. For a given sublattice spacing b, any point in $\mathcal{L}(\mathbf{I})$ lies in one of b^d affine sublattices $\mathcal{L}(b\mathbf{I})_{\mathbf{c}}$. These sublattices can be said to span $\mathcal{L}(\mathbf{I})$. An example of this can can be seen in Figure 3.11, where the 6x6 lattice is spanned by 3^2 affine sublattices of spacing 3.



(a) Sublattices with offsets $\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}2\\1\end{pmatrix}, \begin{pmatrix}1\\2\end{pmatrix}$ (b) Sublattices with offsets $\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}2\\2\end{pmatrix}$ (c) Sublattices with offsets $\begin{pmatrix}2\\0\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix}$

Figure 3.11: The decomposition of a 6x6 lattice into 3^2 sublattices $\mathcal{L}(3I)_{\mathbf{c}_0}$.

More formally, given $b \in \mathbb{Z}$, the sublattices that span the entire lattice $\mathcal{L}(\mathbf{I})$ are:

$$\mathcal{L}(b\mathbf{I})_{\mathbf{c}_i} = \mathcal{L}(b\mathbf{I}) + \mathbf{c}_i, \text{ with } \mathbf{c}_0 = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \mathbf{c}_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \ldots, \mathbf{c}_{b^d-1} = \begin{pmatrix} b-1 \\ b-1 \\ \vdots \\ b-1 \end{pmatrix}. \quad (3.13)$$

As there are b distinct options for each of the d elements of an offset $\mathbf{c}$, there are b^d distinct lattice bases that span $\mathcal{L}(\mathbf{I})$. Based on the b-radix representation of integers, we can find a one to one function that maps the integers $0 \le i \le b^d - 1$ to each offset vector $\mathbf{c}$, allowing each $\mathbf{c}$ to be associated with a unique sublattice number. The function that maps $\mathbf{c}$ to i is

$$i = \sum_{j=1}^{d} c_j b^{j-1}. \quad (3.14)$$

Its inverse function that maps i to a particular offset $\mathbf{c}$ is computed by Algorithm 5. This gives the following general equation for the i-th affine sublattice basis

$$\mathcal{L}(b\mathbf{I})_{\mathbf{c}_i} = \mathcal{L}(b\mathbf{I}) + \begin{pmatrix} \lfloor \frac{r_{d-1}}{b^0} \rfloor \\ \vdots \\ \lfloor \frac{r_1}{b^{d-2}} \rfloor \\ \lfloor \frac{i}{b^{d-1}} \rfloor \end{pmatrix}. \quad (3.15)$$

Algorithm 5 $\mathbf{c}$ = ConvertIndexToOffset(i, b, d)
% Find the affine offset $\mathbf{c}$, given its integer reference number i
% Input: i: integer lattice reference
% Output: The offset vector $\mathbf{c}$

```
1: for $m = d \to 1$ do
2:     $\mathbf{c}(m) \leftarrow \lfloor \frac{i}{b^{m-1}} \rfloor$
3:     $\mathbf{r}_m \leftarrow i(\text{mod } b^{m-1})$
4:     $i \leftarrow \mathbf{r}_m$
5: end for
       return c
```

Because the sublattices span the lattice, the coordinates of any lattice point $\mathbf{x}$ can be represented as $x_i * b + c_i, x_i, c_i \in \mathbb{Z}, 0 \le c_i \le (b-1)$, or $b\mathbf{x} + \mathbf{c}$ for some offset vector $\mathbf{c}$ in (3.13). Therefore, taking each coordinate mod b yields the offsets $\mathbf{c} = (c_i)$, which determine through (3.14) which sublattice the point lies in. Consider the example in Figure 3.12. The top left sublattice, consists of the points $(0,0)$, $(0,3)$,$(3,0)$,$(3,3) \equiv (0,0) \pmod 3$. From (3.14), $i = 0*b^1+0*b^0 = 0$, indicating these points are in the 0-th sublattice. Alternatively, the points $(2,1)$,$(5,1)$,$(2,4)$,$(5,4) \equiv (2,1) \pmod 3$. Since $i = 1*3+2 = 5$, these points are in the 5-th sublattice. Finally, points $(1,2)$,$(1,5)$,$(4,5)$,$(4,2) \equiv (1,2) \pmod 3$, which means these points are in the 7-th sublattice ($i = 2*3+1 = 7$).

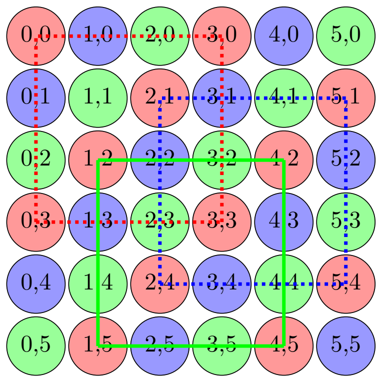


Figure 3.12: Affine sublattices with x,y coordinates.

Consider now a finite lattice with d dimensions of length $d_i, i = 1, \ldots, d$. Let $F_i = factor(d_i)$ be the sorted list of the integers resulting from the prime factorization of d_i. Then define the list of common factors as $\mathbf{F} = sort(\bigcap_{i=1}^{d} F_i)$. In the example of Figure 3.11, $F_1 = F_2 = \{2,3\}$, and so $\mathbf{F} = \{2,3\}$. More interestingly, consider a lattice of dimensions 60×140. Then $F_1 = \{2,2,3,5\}, F_2 = \{2,2,5,7\}$, and $\mathbf{F} = \{2,2,5\}$.

We can use the list of common factors $\mathbf{F}$ to split the lattice $\mathcal{L}(\mathbf{I})$ into a hierarchy of spanning sublattices. We start with the smallest $b = \mathbf{F}(1)$ and obtain the sublattices in (3.13). Then we split every sublattice into its own set of spanning sublattices based on the next common factor $\mathbf{F}(2)$. The process continues recursively until all common factors have been exhausted. Using the fact that for any point $\mathbf{p}$ in a lattice, its sublattice offset after a split is ($\mathbf{p} \bmod b$), Algorithm 6 computes the sublattice offset vectors of $\mathbf{p}$ for all levels. Because of the equivalence between offsets and indices, Algorithm 6 returns only the index of the offsets through (3.14). Note that after splitting $\mathcal{L}(\mathbf{I})$ with $b = \mathbf{F}(1)$, the

$\mathcal{L}(b\mathbf{I})_{\mathbf{c}_i}$ have common factors $\mathbf{F}(2:\mathit{end})$, and all have the same size with dimensions d_i/b. The coordinates of $\mathbf{p}$ in its sublattice are $\lfloor \frac{\mathbf{p}}{b} \rfloor$.

Algorithm 6 $[i^{(1)}, \ldots, i^{(f)}] = \text{SublatticeIndicesOfPoint}(\mathbf{p}, \mathbf{F})$

```
% Determine which sublattice a point lies in at each splitting level
% Input: $\mathbf{p}$ lattice point coordinates
% Input: $\mathbf{F}$ the common prime factors $\mathbf{F}(1) \leq \cdots \leq \mathbf{F}(f)$
% Output: $i^{(m)}$ the index corresponding to offset $\mathbf{c}^m, m = 1, \ldots, f$ of the $m$-th split
1: for $m = 1 \to size(\mathbf{F})$ do          % At each level use splitting spacing $b = \mathbf{F}(m)$
2:     $\mathbf{c}^m \leftarrow \mathbf{p}(\text{mod } \mathbf{F}(m))$     % determine $\mathbf{p}$'s affine offset (i.e., sublattice) at level $m$
3:     $i^{(m)} \leftarrow$ Convert $\mathbf{c}^m$ to index through (3.14)
4:     $\mathbf{p} \leftarrow \lfloor \mathbf{p}/\mathbf{F}(m) \rfloor$          % point coordinates in this sublattice
5: end for
       return $i^{(m)}$ for all $m$
```

3.4 Coloring sublattices

The distance between any two points $\mathbf{p}_1, \mathbf{p}_2 \in \mathcal{L}(b\mathbf{I})_{\mathbf{c}}$ is $\|\mathbf{p}_1 - \mathbf{p}_2\|_1$. The minimum distance of these points is b, the spacing of the sublattice. More formally, from (3.12), there exist $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{Z}^d$, such that $\mathbf{p}_1 = b\mathbf{x}_1 + \mathbf{c}, \mathbf{p}_2 = b\mathbf{x}_2 + \mathbf{c}$, and, if $\mathbf{p}_1, \mathbf{p}_2$ are unique, $\|\mathbf{p}_1 - \mathbf{p}_2\|_1 = b\|\mathbf{x}_1 - \mathbf{x}_2\|_1 \geq b$. Thus, we may assign the same color in all points in $\mathcal{L}(b\mathbf{I})_{\mathbf{c}}$ and still have a valid distance $b-1$ coloring of the points within $\mathcal{L}(b\mathbf{I})_{\mathbf{c}}$.

However, the minimum distance between points in two different sublattices is determined by the distance of their offsets. Using (3.12) again, if $\mathbf{p}_1 \in \mathcal{L}(b\mathbf{I})_{\mathbf{c}_i}$ and $\mathbf{p}_2 \in \mathcal{L}(b\mathbf{I})_{\mathbf{c}_j}$, $\|\mathbf{p}_1 - \mathbf{p}_2\|_1 = \|b(\mathbf{x}_1 - \mathbf{x}_2) + \mathbf{c}_i - \mathbf{c}_j\|_1 \geq \|\mathbf{c}_i - \mathbf{c}_j\|_1$, since we can pick $\mathbf{x}_1 = \mathbf{x}_2$. For example, points $[0, \ldots, 0]^T \in \mathcal{L}(b\mathbf{I})_{\mathbf{c}_0}$ and $[1, 0, \ldots, 0]^T \in \mathcal{L}(b\mathbf{I})_{\mathbf{c}_1}$ are distance 1 apart. Therefore, if the nodes in $\mathcal{L}(b\mathbf{I})_{\mathbf{c}_0}$ and in $\mathcal{L}(b\mathbf{I})_{\mathbf{c}_1}$ are all assigned the same color, we cannot achieve a valid distance 1 coloring on the entire $\mathcal{L}(\mathbf{I})$.

The problem is equivalent to coloring the finite toroidal lattice

$$\mathcal{C} = \{\mathbf{p} \bmod b,\ \mathbf{p} \in \mathcal{L}(\mathbf{I})\}, \tag{3.16}$$

whose points are the b^d offset vectors $\mathbf{c}_i$ in (3.13). Note that $\mathcal{C}$ can be used to tile the original lattice as seen in Figure 3.13. Different coloring strategies of $\mathcal{C}$ achieve different distances between $\mathbf{c}_i, \mathbf{c}_j$ with the same color, and hence between the points of $\mathcal{L}(b\mathbf{I})_{\mathbf{c}_i}, \mathcal{L}(b\mathbf{I})_{\mathbf{c}_j}$. More formally we have the following.

Lemma 3.5 *Assume that each $\boldsymbol{p} \in \mathcal{L}(\boldsymbol{I})$ is assigned a color, color$(\boldsymbol{p})$, and that all points in each $\mathcal{L}(b\boldsymbol{I})_{\boldsymbol{c}}$ are assigned the same color, i.e., $\forall \boldsymbol{p}_i, \boldsymbol{p}_j \in \mathcal{L}(b\boldsymbol{I})_{\boldsymbol{c}}$, color$(\boldsymbol{p}_i) =$ color$(\boldsymbol{p}_j)$. Then color$(\boldsymbol{p} \bmod b) =$ color$(\boldsymbol{p})$.*

Proof: *Since $\cup_{\boldsymbol{c}} \mathcal{L}(b\boldsymbol{I})_{\boldsymbol{c}} = \mathcal{L}(\boldsymbol{I})$, $\boldsymbol{p} = b\boldsymbol{x} + \boldsymbol{c}$. Then color$(b\boldsymbol{x} + \boldsymbol{c} \bmod b) =$ color$(\boldsymbol{c})$, and since $p, c \in \mathcal{L}(b\boldsymbol{I})_{\boldsymbol{c}}$, both have the same color.* ■

To take advantage of the b spacing of the points within each $\mathcal{L}(b\mathbf{I})_{\mathbf{c}}$, one obvious strategy is to assign every $\mathbf{c}_i$ in $\mathcal{C}$ (equivalently each sublattice) a unique color. This guarantees a distance $b-1$ coloring for the entire lattice $\mathcal{L}(\mathbf{I})$. In the context of our recursive splitting algorithm, the first split with $b_1 \in \mathbf{F}$ uses b_1^d colors, and achieves $b_1 - 1$ distance coloring. At the second recursive level with $b_2 \in \mathbf{F}$, each $\mathcal{L}(b_1\mathbf{I})_{\mathbf{c}}$ is split into b_2^d sublattices, each with a unique color, for a total of $(b_1 b_2)^d$ colors. Points in $\mathcal{L}(b_2\mathbf{I})_{\mathbf{c}}$ are at least b_2 hops apart, but these hops are edges in the $\mathcal{L}(b_1\mathbf{I})_{\mathbf{c}}$ lattice. Thus the minimum distance achieved by this coloring at the second level is $b_1 b_2 - 1$. A simple inductive argument shows the following.

Lemma 3.6 *If at every level of the recursive splitting algorithm each sublattice is assigned a unique color, then at level m we have used $(b_1 \cdots b_m)^d$ colors and have achieved a distance $b_1 \cdots b_m - 1$ coloring.*

The algorithm increases the effective distance exponentially with each level, and probing with the corresponding vectors should be very effective. However, the number of colors (and of probing vectors) used increases rapidly too. This is not an efficiency problem but rather an evaluation problem. After level $m-1$, probing cannot fully annihilate elements of distance $b_1 \cdots b_m - 1$ until all $(b_1 \cdots b_m)^d$ colors have been used. Thus, we cannot properly evaluate its progress for intermediate numbers of colors. Moreover, the number of probing vectors in the next level may not be affordable computationally. For example, if $b_i = 2$ and $d = 4$, we can only guarantee meaningful results at color numbers, $16, 256, 4096, \ldots$. Therefore, it would be desirable to maintain the effectiveness of the method when each node in $\mathcal{C}$ is uniquely colored, but also have one or more intermediate evaluation points where smaller distance colorings complete.

A problem with more than one intermediate points is that the colorings of $\mathcal{C}$ at two different distances must be hierarchical, i.e., two lattice nodes that have different colors at distance j can not have the same color at larger distance colorings. Also, to facilitate the generation of probing vectors on-demand, each color should have the same number of nodes (see later discussion). Since b is prime, we can only consider colorings with b, or $b^2, \ldots$, or b^d colors. For example, a red-black coloring of $\mathcal{C}$ is not a valid distance 1 coloring for any odd b. The periodic connection links two nodes (and thus sublattices) with the same color. We will see experimentally that the errors from ignoring these connections can be significant. Instead, three colors can provide a valid distance 1 coloring of any toroidal lattice [66]. However, for $b \neq 3$, the three color subsets would not have the same number of nodes.

We are not aware of a method that produces optimal distance colorings of $\mathcal{C}$ for any b and d. For small values of b, d we have identified heuristics that using b colors produce a valid distance $O(b^{1/d})$ coloring of $\mathcal{C}$. For practical problems as in LQCD, $d \leq 4$ and $b \leq 7$, so the effective distance achieved is not better than distance 1. Besides, an optimal coloring is not necessary as the nodes in the same colors will be hierarchically colored too so that nearby permuted nodes eventually have large distances. Therefore, we focus our attention on the following simpler method as our only intermediate point between two recursive levels. This coloring strategy produces a valid distance 1 coloring with b colors, and ensures that each color appears the same number of times. If $\mathbf{p} \in \mathcal{L}(\mathbf{I})$, with

$\mathbf{p} \bmod b = \mathbf{c}$, define its color:

$$\mathit{color}(\mathbf{p}) = \sum_{i=1}^{d} \mathbf{p}(i) \bmod b. \tag{3.17}$$

Because of Lemma 3.5, we also have $\mathit{color}(\mathbf{p}) = \mathit{color}(\mathbf{c}) = \sum_{i=1}^{d} \mathbf{c}(i) \bmod b$. Note that when we reorder the nodes of $\mathcal{C}$ based on this coloring, we also consider the sublattices $\mathcal{L}(b\mathbf{I})_{\mathbf{c}}$ in the same order.

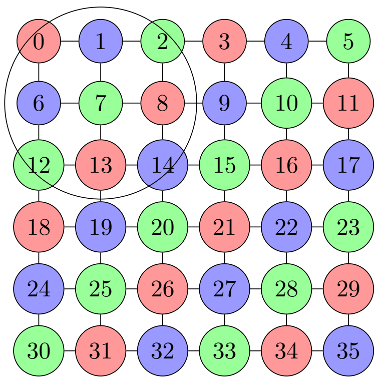


Figure 3.13: The circled nodes constitute the $\mathcal{C}$ lattice of offsets. Note how $\mathcal{C}$ tiles the entire lattice, and that its coloring reflects the coloring of each sublatice $\mathcal{L}(b\mathbf{I})_{\mathbf{c}}$. Since $b = 3$, each line of colors is the same as the previous line, shifted by 1 mod 3.

Lemma 3.7 *The strategy (3.17) is a valid distance-1 coloring of $\mathcal{L}(\boldsymbol{I})$.* **Proof:** *Let $\boldsymbol{p}_1, \boldsymbol{p}_2 \in \mathcal{L}(\boldsymbol{I})$, with $\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|_1 = 1$. This means they share all but one coordinate, say the i-th. If their connection is not due to the periodic boundary, their i-th coordinate will differ by one. Thus, $|\boldsymbol{p}_1(i) - \boldsymbol{p}_2(i)| \bmod b = 1$, implying that $\mathit{color}(\boldsymbol{p}_1) \neq \mathit{color}(\boldsymbol{p}_2)$. If both points lie on a boundary and connect via the toroidal property, then $|\boldsymbol{p}_1(i) - \boldsymbol{p}_2(i)| = (b-1) - 0 \bmod b \not\equiv 0 \bmod b$, and therefore $\mathit{color}(\boldsymbol{p}_1) \neq \mathit{color}(\boldsymbol{p}_2)$. Since these are no other cases possible, the result holds.* ■

The coloring strategy (3.17) has an efficient recursive implementation. Note that the colors in the i-th $(d-1)$-dimensional slice of $\mathcal{C}$ are the colors of the $(i-1)$-th $(d-1)$-dimensional slice shifted by 1 mod b. This can be seen in Figure 3.13 for $d = 2$. Since the

0-th slice is the same as the coloring of $\mathcal{C}$ in $d-1$ dimensions, we can build the colorings for all dimensions in the following recursive manner.

Let $c_{d,b}$ be the array[1] of all b^d colors of the corresponding d-dimensional $\mathcal{C}$ in natural ordering. This $\mathcal{C}$ has b $(d-1)$-dimensional slices each corresponding to the $(d-1)$-dimensional lattice of offsets. Let $c_{d-1,b}$ be the array of the b^{d-1} colors of these $(d-1)$-dimensional lattices. Then, $c_{d,b}$ is a concatenation of b shifted $c_{d-1,b}$ arrays,

$$c_{d,b} = \{c_{d-1,b},\ c_{d-1,b} + 1 \bmod p,\ \dots\ , c_{d-1,b} + (b-1) \bmod p\}\,. \tag{3.18}$$

Each shift applies to all the elements of the array $c_{d-1,b}$. In Figure 3.13, for example, $c_{1,3} = \{0,1,2\}$ are the colors of a one dimensional lattice, but also the first row of the two dimensional $\mathcal{C}$. The colors of $\mathcal{C}$ are then $c_{2,3} = \{\{0,1,2\},\{0,1,2\} + 1 \bmod 3, \{0,1,2\} + 2 \bmod 3\} = \{0,1,2,1,2,0,2,0,1\}$.

Algorithm 7 implements this recursive coloring of $\mathcal{C}$ starting with $c_{0,b} = 0$, and then generates the permutation, Perm, that reorders sublattices of the same color together. Since there is the same number of sublattices for each of the b colors, the first sublattice of color i is assigned to the ib^{d-1} location, and the index to the next available free spot for this color stored in ColorIndex is incremented by one. We note that this coloring occurs negligible computational cost, even for very large lattices, while it enables an additional intermediate step between splits where the trace estimation may be monitored. This low cost is an advantage over other colorings that could be used to define different intermediate steps.

We now have a method that at each level $m = 1, \dots, f$ recursively splits a sublattice into $\mathbf{F}(m)^d$ sublattices, giving each a different color. But before each sublattice is assigned its own color at level m, we have an intermediate coloring that groups together $\mathbf{F}(m)^{d-1}$ sublattices in the same color. According to Lemma 3.6, after the intermediate step before level m we have $(b_1 \cdots b_{m-1})^d b_m$ colors ensuring a distance $b_1 \cdots b_{m-1}$ coloring, and after

[1]The notation of this array is not to be confused with the notation of offsets $\mathbf{c}$ which are in bold.

Algorithm 7 Perm$(0, \ldots, b^d - 1)$ = GenOffsetPermutation(b, d)
% Generate the b-coloring permutation of $\mathcal{C}$ which reorders the sublattices (offsets)
% Input: prime factor b, lattice dimension d
% Output: Perm, the b-color permutation of the b^d sublattices

```
 1: % Generate the coloring using (3.18)
 2: $c_{0,b} \leftarrow \{0\}$
 3: **for** $j = 1 \to d$ **do**
 4:     $c_{j,b} \leftarrow \{c_{j-1,b},\ c_{j-1,b} + 1 \bmod b,\ \ldots,\ c_{j-1,b} + b - 1 \bmod b\}$
 5: **end for**
 6: % Initialize index showing where the next sublattice of color $i$ should go
 7: **for** $i = 0 \to b - 1$ **do**
 8:     ColorIndex(i) $\leftarrow i * b^{d-1}$
 9: **end for**
10: **for** $i = 0 \to b^d - 1$ **do**
11:     Color $\leftarrow c_{d,b}(i)$                % Lookup the color of sublattice $i$ in array $c_{d,b}$
12:     Perm($i$) $\leftarrow$ ColorIndex(Color)         % The new location of sublattice $i$
13:     ColorIndex(Color) $\leftarrow$ ColorIndex(Color) +1
14: **end for**
        **return** Perm
```

the m level we have $(b_1 \cdots b_m)^d$ colors for a distance $b_1 \cdots b_m - 1$ coloring. Next, we describe the global hierarchical permutation and in particular how to find the location in this permutation of an arbitrary node. This will be used to efficiently generate the hierarchical probing vectors.

3.4.1 Hierarchical Permutations of Lattices with Equal Sides

The final permutation can be obtained recursively by applying the coloring permutation from level m on the permuted index from level $m-1$. This ordering ensures that the closer the nodes are geometrically, the farther they are ordered in the permutation. Orderings of lower levels provide no additional information, since nodes never move closer together in subsequent levels, so need not be stored. Moreover, we can avoid the above recursion by determining directly where in the final permutation a node will lie.

Consider the example in Figure 3.11. At level 0, the 3-coloring of the lattice permutes the colors in three groups as shown below on the left.

Level 0, after 3-coloring		Level 1, after splitting to 3^2 sublattices			
color 0:	0 3 8 11 13 16 18 21 26 29 31 34	offsets 0,5,7:	0 3 18 21	8 11 26 29	13 16 31 34
color 1:	1 4 6 9 14 17 19 22 24 27 32 35	offsets 1,3,8:	1 4 19 22	6 9 24 27	14 17 32 35
color 2:	2 5 7 10 12 15 20 23 25 28 30 33	offsets 2,4,6:	2 5 20 23	7 10 25 28	12 15 30 33

At level 1, since $b = 3$, we split the top level lattice to 9 sublattices. Three of those lattices (offsets 0, 5, 7) contain only the red nodes from the intermediate coloring and need to be ordered first. Notice how the actual permutation of the red nodes at level 0 is not needed as it is present in the ordering of the sublattices by Algorithm 7. An exception would be after the last level, if we decide to 3-color the remaining sublattices that do not share common prime factors (as in our previous work [66]).

Algorithm 8 shows we can construct the final hierarchical permutation. Given the coordinates of a node, $\mathbf{p}$, Algorithm 6 generates the indices $[i^{(1)}, \ldots, i^{(f)}]$ of the sublattices the $\mathbf{p}$ lies in at each level. Algorithm 7 permutes these to $[\hat{i}^{(1)}, \ldots, \hat{i}^{(f)}] = \text{Perm}([i^{(1)}, \ldots, i^{(f)}])$. Because the permutation preserves the hierarchy, the location of the node $\mathbf{p}$ would be determined by all the nodes that belong to sublattices that appear prior to its own sublattices in the final permutation. For example, there are exactly $\hat{i}^{(1)}$ sublattices preceding it at the first level, $\hat{i}^{(2)}$ sublattices preceding it at the second level, and so on. At every level m the size of each sublattice reduces by a factor of $F(m)^d$. Thus, given the lattice size $L = \prod_{i=1}^{d} d_i$, we have

$$Location(\mathbf{p}) = \sum_{m=1}^{f} \hat{i}^{(m)} \frac{L}{\prod_{l=1}^{m} \mathbf{F}(l)^d}. \tag{3.19}$$

3.4.2 Hierarchical Permutations of Lattices with Unequal Sides

Algorithm 8 relies on having each lattice split into an equal number of sublattices of the same dimensionality. However, it is possible that one dimension of the lattice may be smaller than the others, leading to that dimension being exhausted before the others. If the rest of the dimensions share factors, the algorithm can continue but in a lower dimensionality lattice that removes exhausted dimensions. Let $\bar{d}^{(m)}$ be the number of active dimensions at level m. For example, the lattice $6 \times 6 \times 2$, has factors $(2, 3), (2, 3), (2)$,

Algorithm 8 Location = HPpermutation($\mathbf{p}, d, \mathbf{F}$)
% Compute the Hierarchical Probing permutation of a node $\mathbf{p}$ when $d_1 = \cdots = d_d$
% Input: point coordinates $\mathbf{p}$, lattice dimension d, common prime factors $\mathbf{F}$
% Output: the location of $\mathbf{p}$ in the HP permutation

```
1: $[i^{(1)}, \ldots, i^{(f)}]$ = SublatticeIndicesOfPoint($\mathbf{p}, \mathbf{F}$)        (Algorithm 6)
2: subLatticeSize = $\prod_{i=1}^{d} d_i$
3: Location = 0
4: for $m = 1 \to f$ do
5:     Perm = GenOffsetPermutation($\mathbf{F}(m), d$)        (Algorithm 7)
6:     subLatticeSize = subLatticeSize/$\mathbf{F}(m)^d$
7:     Location = Location + Perm($i^{(m)}$)*subLatticeSize
8: end for
       return Location
```

so for the first level $\bar{d}^{(1)} = 3$, while for the second level $\bar{d}^{(2)} = 2$, since at the second level all sublattices will be 2 dimensional. Thus, $\bar{d}^{(m)}$ can be computed simply by counting the number of common factors in each dimension that has not been exhausted. Then, the new location of a node is given by (3.20), [2]

$$Location(\mathbf{p}) = \sum_{m=1}^{f} \hat{i}^{(m)} \frac{L}{\prod_{l=1}^{m} \mathbf{F}(l)^{\hat{d}^{(m)}}}. \tag{3.20}$$

We can avoid computing and storing coloring permutations for lattices with reducing dimensionalities by reusing the previously computed permutations for $\mathcal{C}$ of a higher dimensionality in (3.16), as long as the spacing b is the same. First, recall that for a given $\mathcal{C}$ of dimensionality d and spacing b, the coloring $c_{d,b}$ in (3.18) is created recursively. This means that the color of a particular node in $\mathcal{C}$ can be given in terms of either a higher or a lower dimensional $\mathcal{C}$ plus a correctional offset as below,

$$c_{d,b}(k) = c_{d-1,b}(k \bmod b^{d-1}) + \lfloor \frac{k}{b^{d-1}} \rfloor \bmod b, \ \forall k < b^d, \tag{3.21}$$

$$c_{d-1,b}(k \bmod b^{d-1}) = c_{d,b}(k) - \lfloor \frac{k}{b^{d-1}} \rfloor \bmod b, \ \forall k < b^d. \tag{3.22}$$

[2]It is worth noting that just as [66] interpreted this process as representing the node number in binary and then permuting the digits, we can represent each node in mixed radix, where the radix list is the color numbers used to color the sublattices at each level, and then permute these digits.

Additionally, we shall make use of the definition of mod for positive integers

$$k \bmod b = k - b\lfloor \frac{k}{b} \rfloor. \tag{3.23}$$

Lemma 3.8 *For any prime b, $c_{d,b}(ib) = c_{d-1,b}(i)$, $\forall i = 0, \ldots, b^{d-1} - 1$.*

Proof: We proceed by induction on d. For the base case, $c_{1,b} = \{0 \ldots b-1\}$, and $c_{2,b} = \{c_{1,b}, c_{1,b} + 1 \bmod b, \ldots, c_{1,b} + b - 1 \bmod b\}$. Then by construction, every $c_{2,b}(ib) = 0 + i = c_{1,b}(i)$. Assume that $c_{d-1,b}(ib) = c_{d-2,b}(ib/b) = c_{d-2,b}(i)$. Then,

$$\begin{array}{rcll}
c_{d,b}(ib) & = & c_{d-1,b}(ib \bmod b^{d-1}) + \lfloor \frac{ib}{b^{d-1}} \rfloor \bmod b & (\text{ by } 3.21) \\
& = & c_{d-2,b}(\frac{ib \bmod b^{d-1}}{b}) + \lfloor \frac{ib}{b^{d-1}} \rfloor \bmod b & (\text{by the I.H.}) \\
& = & c_{d-2,b}(\frac{ib - \lfloor \frac{ib}{b^{d-1}} \rfloor b^{d-1}}{b}) + \lfloor \frac{ib}{b^{d-1}} \rfloor \bmod b & (\text{by } 3.23) \\
& = & c_{d-2,b}(i \bmod b^{d-2}) + \lfloor \frac{ib}{b^{d-1}} \rfloor \bmod b & (\text{by } 3.23) \\
& = & c_{d-1,b(i)} - \lfloor \frac{i}{b^{d-2}} \rfloor + \lfloor \frac{ib}{b^{d-1}} \rfloor \bmod b & (\text{by } 3.22) \\
& = & c_{d-1,b(i)} - \lfloor \frac{i}{b^{d-2}} \rfloor + \lfloor \frac{i}{b^{d-2}} \rfloor \bmod b & \\
& = & c_{d-1,b(i)}. &
\end{array}$$

■

Lemma 3.9 *For any prime b, $c_{d,b}(ib) = c_{d,b}(ib+q) - q \bmod b$, $\forall i = 0, \ldots, b^{d-1} - 1, \forall q = 0, \ldots, b-1$.*

Proof: We proceed by induction on d. For the base case, by construction we have $c_{2,b}(ib+q) \bmod b = c_{1,b}(i) + q \bmod b$. Then, $c_{2,b}(ib+q) - q \bmod b = (c_{1,b}(i) + q \bmod b - q) \bmod b = c_{1,b}(i) \bmod b = c_{1,b}(i) = c_{2,b}(ib)$ (by Lemma 3.8). We now assume $c_{d-1,b}(ib) = c_{d-1,b}(ib + q \bmod b^{d-1}) - q \bmod b$. Then,

$$\begin{array}{rcll}
c_{d,b}(ib) & = & c_{d-1,b}(ib \bmod b^{d-1}) + \lfloor \frac{ib}{b^{d-1}} \rfloor \bmod b & (\text{by } 3.21) \\
& = & c_{d-1,b}(ib + q \bmod b^{d-1}) - q + \lfloor \frac{ib}{b^{d-1}} \rfloor \bmod b & (\text{by the I.H.}) \\
& = & c_{d,b}(ib + q \bmod b^{d}) - q \bmod b & (\text{by } 3.22) \\
& = & c_{d,b}(ib + q) - q \bmod b. & (\text{since } ib + q < b^d)
\end{array}$$

■

Lemma 3.10 *Let* Perm_d *be the permutation created by Algorithm 7 associated with dimension d. Then, for any prime b,* $i = 0, \ldots, b^d - 1$, $Perm_d(i) = \lfloor i/b \rfloor + c_{d,b}(i) b^{d-1}$.

Proof: Because of Lemma 3.9 when any b-tuple of indices $(bi, bi+1, \ldots, bi+b-1)$, is considered, the number of nodes in every color increases by 1. Since Algorithm 7 will send the b-th color to the b^{d-1}-th section, the equation holds. ■

Theorem 3.4 *For any* $0 < m < d$, Perm_m *can be obtained directly as follows,*

$$\mathrm{Perm}_m(i) = \lfloor \mathrm{Perm}_d(ib^{d-m})/b^{d-m} \rfloor, \ i = 0, \ldots, b^m - 1.$$

Proof: Since $i \leq b^m$, we can apply Lemma 3.8 recursively,

$$c_{d,b}(ib^{d-m}) = c_{d-1,b}(ib^{d-m-1}) = c_{d-2,b}(ib^{d-m-2}) = \ldots = c_{m,b}(i).$$

Using this and Lemma 3.10 we have

$$\begin{aligned}\left\lfloor \frac{\mathrm{Perm}_d(ib^{d-m})}{b^{d-m}} \right\rfloor &= \left\lfloor \frac{\lfloor \frac{ib^{d-m}}{b} \rfloor + c_{d,b}(ib^{d-m}) b^{d-1}}{b^{d-m}} \right\rfloor = \left\lfloor \frac{i}{b} + c_{m,b}(i) b^{m-1} \right\rfloor \\ &= \lfloor \frac{i}{b} \rfloor + c_{m,b}(i) b^{m-1} = \mathrm{Perm}_m(i).\end{aligned}$$

■

Based on Theorem 3.4, Algorithm 9 shows how to reuse previously generated $c_{d,b}$ to compute permutations for lower dimensional lattices when the smaller lattice dimensions are exhausted.

Algorithm 9 Location = HPpermutation_general($\mathbf{p}, d, d_i, \mathbf{F}$)
% Compute the Hierarchical Probing permutation of a node $\mathbf{p}$ for general d_i
% Input: point coordinates $\mathbf{p}$, lattice dimension d, common prime factors $\mathbf{F}$
% Output: the location of $\mathbf{p}$ in the HP permutation

```
 1: $[i^{(1)}, \ldots, i^{(f)}]$ = SublatticeIndicesOfPoint($\mathbf{p}, \mathbf{F}$)        % Algorithm 6
 2: subLatticeSize = $\prod_{i=1}^{d} d_i$
 3: Location = 0
 4: for $m = 1 \to f$ do
 5:     $\hat{d}^{(m)}$ = setActiveDims()
 6:     maxBdim = 0
 7:     for $i = d$ downto $\hat{d}$ do       % Search and retrieve highest dimensional
 8:         Perm = getHash($b, i$);      % stored permutation for a $b$ split
 9:         if Perm != empty then  % if found one, reuse it
10:             maxBdim = i;
11:             break
12:         end if
13:     end for
14:     if maxBdim then
15:         for $i = 1 \to b^{\hat{d}^{(m)}}$ do
16:             newperm(i) = Perm($ib^{maxBdim-\hat{d}^{(m)}}/b^{maxBdim-\hat{d}^{(m)}}$) % Theorem 3.4
17:         end for
18:         Perm = newperm
19:     else
20:         Perm = GenOffsetPermutation($\mathbf{F}(m), d$)        % Algorithm 7
21:     end if
22:     setHash($b, d$) = Perm;      % Store this permutation
23:     subLatticeSize = subLatticeSize/$\mathbf{F}(m)^{\hat{d}^{(m)}}$          % Equation (12)
24:     Location = Location + Perm($i^{(m)}$)*subLatticeSize
25: end forreturn Location
```

3.4.3 Generating Probing Vectors Quickly

In [66], we introduce the following recursive method for generating probing vectors for a colored lattice,

$$\tilde{Z}_{(1)} = \mathcal{F}_{z(1)}, \tag{3.24}$$

$$\tilde{Z}_{(i)} = \left[\tilde{Z}_{(i-1)} \otimes \mathcal{F}_{z(i)}(:,1), \ldots, \tilde{Z}_{(i-1)} \otimes \mathcal{F}_{z(i)}(:,z(i))\right], \tag{3.25}$$

$$Z_{(i)} = \tilde{Z}_{(i)} \otimes \mathbf{1}_{N/\gamma_i}, \quad \text{where } \gamma_i = \prod_{j=1}^{i} z(j). \tag{3.26}$$

Here $\mathcal{F}_{z(i)}$ is the Fourier transform of the identity matrix $I_{z(i)}$, $z(i)$ is the number of colors each sublattice is split into at level i, $\mathbf{1}_s$ is the vector of s ones, and $\otimes$ is the Kronecker product. Essentially, these vectors recursively build a basis for the probing vectors. At each level, we probe inside each color (i.e., sublattice) by smaller probing vectors hierarchically, which are all assembled into a basis through the Kronecker products. Instead of generating the whole matrix, however, we produce each probing vector one at a time, hence requiring the same memory as the Hutchinson method.

To produce the k-th vector of the probing matrix, we first need to identify the maximum level i needed such that $\gamma_{i-1} < k \leq \gamma_i$. Then at every lower recursive level of (3.25) we only need two vectors; one vector from $Z_{(i-1)}$ and one from $\mathcal{F}_{z(i)}$. By (3.25), the matrix $Z_{(i)}$ is divided into $z(i)$ blocks, with each block forming a Kronecker product with a different column of $\mathcal{F}_{z(i)}$. Since each block has $z(i-1)$ columns, the k-th vector is in $block = \lfloor \frac{k}{z(i-1)} \rfloor$, and thus we can generate directly the desired column $\mathcal{F}_{z(i)}(:, block)$. This should be paired with the $(k \bmod z(i-1))$ vector of $Z_{(i-1)}$ which we find recursively with the above procedure.[3]

When $\mathbf{F}(i) = 2,\ i = 1, \ldots, k$, the sublattices in the first k levels are red-black colorable, and thus use only $\mathcal{F}_2$. Because $\mathcal{F}_2$ is equal to the Hadamard matrix H_2, all vectors in the

[3]We note that this process can also be described in terms of radix conversion. Let the $z(i)$s be taken as the radix list. If k is converted to this mixed radix form [**?**], the vectors of the Fourier transforms $\mathcal{F}_{z(i)}$ needed at each level will be the digits of this representation.

first k levels can be created using real arithmetic, which yields substantial savings over complex arithmetic. Moreover, we can use the fast bit-based method we introduced in [66] for producing the required Hadamard vectors, leading to an additional performance gain. This approach can be seen in Algorithm 10.

Algorithm 10 ProbingVector = GenerateProbingVector(k, z, N)
% Compute the k-th probing vector
% Input: k, the number of colors at each level $z(i)$, the matrix size N
% Output: The probing vector

```
1:  for $j = size(z)$ downto 2 do                % Compute indices of needed $\mathcal{F}_{z(i)}$ vectors
2:      $block(j) = \lfloor k/z(i-1) \rfloor$;  $k = k \bmod z(i-1)$
3:  end for
4:  $block(1) \leftarrow k$;
5:  % Find the initial 2-colorable sublattices, which can be probed quickly as in [66]
6:  while $z(j) == 2$ do $j \leftarrow j + 1$; end while
7:  fastLevels $\leftarrow j - 1$
8:  ProbingVector $\leftarrow$ [1]
9:  ProbingVector $\leftarrow$ FastHadamardMethod(1:fastLevels)
10: % The rest of the levels are built through Fourier vectors
11: for $j$ = fastLevels+1 $\rightarrow$ size(z) do
12:     % Create Fourier vectors $\mathcal{F}$
13:     $f \leftarrow 2 * \pi / z(i)$
14:     $w \leftarrow [0 : f : (2 * \pi - f/2)] * \sqrt{-1}$
15:     $\mathcal{F}_{z(i)}(:, block) \leftarrow exp(-w \otimes block(j))$
16:     ProbingVector $\leftarrow$ ProbingVector $\otimes \mathcal{F}_{z(i)}(block, :)$
17: end for
        return ProbingVector
```

Since Algorithm 10 is based on coloring strategy (3.17), the first γ_i probing vectors in (3.26) correspond to what we termed as the intermediate coloring between splitting levels $i-1$ and i. However, there are other numbers k, with $\gamma_{i-1} < k < \gamma_i$, for which $Z(:, 1:k)$ are the probing vectors for a different valid coloring, most importantly the one in Lemma 3.6 where each sublattice gets a unique color. This is because the coloring is hierarchical, i.e., (3.17) is applied independently on each sublattice. Let us return to the example in Figure 3.11, where we first consider the factor $b = 3$ and then $b = 2$. The results after the 3-coloring and the first level sublattice split are shown in Section 3.4.1. Here we focus only on color 0:

Level 0, after 3-coloring

color 0: $\boxed{0\ 3\ 8\ 11\ 13\ 16\ 18\ 21\ 26\ 29\ 31\ 34}$

Level 1, after splitting to 3^2 sublattices

offsets 0,5,7: $\boxed{0\ 3\ 18\ 21}$ $\boxed{8\ 11\ 26\ 29}$ $\boxed{13\ 16\ 31\ 34}$

Level 1, after 2-coloring each sublattice we have a total of 2×3^2 colors

original color 0 now contains $6 = 2 \times 3$ colors $\boxed{0\ 21}$ $\boxed{3\ 18}$ $\boxed{8\ 29}$ $\boxed{11\ 26}$ $\boxed{13\ 34}$ $\boxed{16\ 31}$

Notice therefore that by construction the first nine indices in our final permutation (which is used to generate the probing vectors $Z(:, 1:9)$) correspond to the coloring at level 1 where each lattice has a different color.

3.5 Probing Vectors For Hierarchical Coloring on General Graphs

The above method for generating probing vectors assumes each color splits into the same number of colors at a given level. This is the case with our methods in Section 3.4. With an arbitrary coloring method that does not assign the same number of sublattices to each color (e.g., the 3-coloring method on a lattice not divisible by three), a different way to generate the probing vectors is needed. A simple but not as efficient solution is to create the required canonical probing vectors, and then orthogonalize them against previous vectors in the sublatice as well as each other with Gram-Schmidt. We introduce a more efficient and elegant method that works with uneven color splits and thus generalizes probing to any graph with hierarchical coloring.

The method is described better through an example. Consider a graph with seven nodes (each node could be generalized to be a subgraph). Suppose at the first level the graph is assigned three colors. After the corresponding permutation, the first color contains nodes 1–3, the second color nodes 4 and 5, and the third color nodes 6 and 7. To probe at the first level we use the following probing vectors which is a variation of $\mathcal{F}_3$ in

(3.24) and (3.26) to allow for different number of nodes per color,

$$\mathcal{Z}_{(1,0)} = \begin{bmatrix} \mathcal{F}_3(1,:) \otimes \mathbf{1}_3 \\ \mathcal{F}_3(2,:) \otimes \mathbf{1}_2 \\ \mathcal{F}_3(3,:) \otimes \mathbf{1}_2 \end{bmatrix} \in \mathbb{C}^{7\times 3}. \tag{3.27}$$

Suppose now that at the second level, the first color splits into three colors and the others into two. Clearly, the next level of probing vectors cannot be created by (3.25) because of uneven splitting. Each color block of the first level has to be probed independently. Thus, we could probe the first block using $\mathcal{F}_3$ for the elements inside the first block (with zeros everywhere else in the probing vector). Similarly for the last two blocks, but using $\mathcal{F}_2$. These seven probing vectors are shown in (3.28) —note that $\mathbf{0}_k = \text{zeros}(k,1)$. The problem is that using seven vectors would be wasting the solutions of linear systems with the three probing vectors (3.27) in the first step.

$$\begin{bmatrix} \mathcal{F}_3(:,1) \\ \mathbf{0}_2 \\ \mathbf{0}_2 \end{bmatrix} \begin{bmatrix} \mathcal{F}_3(:,2) \\ \mathbf{0}_2 \\ \mathbf{0}_2 \end{bmatrix} \begin{bmatrix} \mathcal{F}_3(:,3) \\ \mathbf{0}_2 \\ \mathbf{0}_2 \end{bmatrix}, \begin{bmatrix} \mathbf{0}_3 \\ \mathcal{F}_2(:,1) \\ \mathbf{0}_2 \end{bmatrix} \begin{bmatrix} \mathbf{0}_3 \\ \mathcal{F}_2(:,2) \\ \mathbf{0}_2 \end{bmatrix}, \begin{bmatrix} \mathbf{0}_3 \\ \mathbf{0}_2 \\ \mathcal{F}_2(:,1) \end{bmatrix} \begin{bmatrix} \mathbf{0}_3 \\ \mathbf{0}_2 \\ \mathcal{F}_2(:,2) \end{bmatrix}. \tag{3.28}$$

The key to remedying this problem is to note that the three first vectors of the new color blocks,

$$\mathcal{I} = \begin{bmatrix} \mathcal{F}_3(:,1) & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_2 & \mathcal{F}_2(:,1) & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{0}_2 & \mathcal{F}_2(:,1) \end{bmatrix} = \begin{bmatrix} \mathbf{1}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_2 & \mathbf{1}_2 & \mathbf{0}_2 \\ \mathbf{0}_2 & \mathbf{0}_2 & \mathbf{1}_2 \end{bmatrix}$$

are spanned by the vectors of $\mathcal{Z}_{(1,0)}$, since $\mathcal{F}_3$ is a basis of $\mathbb{C}^3$. More formally, if $\mathbf{a} \in \mathbb{C}^{3\times 3}$, from (3.27) and basic properties of the Kronecker product we have that the following matrix equation

$$\mathcal{Z}_{(1,0)}\mathbf{a} = \begin{bmatrix} \mathcal{F}_3(1,:)\mathbf{a} \otimes \mathbf{1}_3 \\ \mathcal{F}_3(2,:)\mathbf{a} \otimes \mathbf{1}_2 \\ \mathcal{F}_3(3,:)\mathbf{a} \otimes \mathbf{1}_2 \end{bmatrix} = \mathcal{I},$$

is equivalent to $\mathcal{F}_3\mathbf{a} = I_3$, which has a unique solution $\mathbf{a} = \text{ifft}(I_3)$, i.e., the inverse Fourier transform of the identity. Therefore, if we saved $P = A^{-1}\mathcal{Z}_{(1,0)}$, we can recover the probing result for the vectors in $\mathcal{I}$ as $A^{-1}\mathcal{I} = P\mathbf{a}$. Thus, we only need to apply A^{-1} on the remaining four probing vectors, exactly as in our hierarchical probing method on the lattice. Finally, if each node represents a subgraph, each color block in (3.28) involves Kronecker products of the rows of its Fourier matrix with columns of ones, each sized to the cardinality of the subgraph. Thus, each block has the same form as (3.27) and the idea can be applied recursively.

To generalize we need the following definitions. First, assume a hierarchical coloring at levels $i = 0, 1, \ldots$, and let l_{i-1} be the number of colors at level $i-1$. The nodes belonging to one of these colors are called a block at the next level i. There are l_{i-1} blocks at the i-th level. Let $s(j, i)$ be the number of colors the j-th block splits into at level i. Thus, $\sum_{j=1}^{l_{i-1}} s(j, i) = l_i$. For each color in block j, let $n(j, i, k),\ k = 1, \ldots, s(j, i)$ be the number of nodes in that color. Thus, $\sum_{k=1}^{s(j,i)} n(j, i, k)$ is the number of nodes in the j-th block. For each $j = 1, \ldots, l_{i-1}$ block, define the Fourier transform $\mathcal{F}_{s(j,i)} = \text{fft}(I_{s(j,i)})$, and $\mathcal{Z}_{(j,i)}$ the set of probing vectors as

$$\mathcal{Z}_{(j,i)} = \begin{bmatrix} \mathbf{0} \\ \mathcal{F}_m(1, :) \otimes \mathbf{1}_{n(j,i,1)} \\ \vdots \\ \mathcal{F}_m(m, :) \otimes \mathbf{1}_{n(j,i,m)} \\ \mathbf{0} \end{bmatrix}, \text{ where } m = s(j, i). \tag{3.29}$$

The $\mathbf{0}$ zero matrices have m columns and rows that overlap with all other blocks. At level 0, there is only one block (with $s(1, 0)$ colors), so the $\mathbf{0}$ matrices are empty. E.g., the first three vectors in (3.28) are $\mathcal{Z}_{(1,1)}$, and $\mathcal{I} = [\mathcal{Z}_{(1,1)}(:, 1), \mathcal{Z}_{(2,1)}(:, 1), \mathcal{Z}_{(3,1)}(:, 1)]$.

We assume that the results of the inversions have been saved $P = A^{-1}\mathcal{Z}_{(j,i-1)}$ for all blocks $j = 1, \ldots, l_{i-2}$ at level $i-1$. At the i-th level, probing with the first vectors of each block can be determined as follows:

$$\begin{aligned} \mathbf{a} &= \text{ifft}(I_{l_{i-1}}), & (3.30)\\ A^{-1}\mathcal{Z}_{(j,i)}(:, 1) &= P\mathbf{a}(:, j),\ j = 1, \ldots, l_{i-1}, & (3.31) \end{aligned}$$

or equivalently note that

$$P\mathbf{a} = \text{ifft}(P^H)^T. \tag{3.32}$$

Systems for the rest of the probing vectors in the blocks are solved explicitly. At the end of level i, we have inversions for all the probing vectors $\mathcal{Z}_{(i)} = [\mathcal{Z}_{(1,i)} \ldots \mathcal{Z}_{(l_{i-1},i)}]$. If further levels are needed, the process continues as described in Algorithm 11. We emphasize that our new method fuses the generation of probing vectors and the solution of linear systems needed for the trace computation. However, Algorithm 11 depicts only the generation of the vectors.

We note that our method requires memory to store the vectors P at the previous level. When generating the P for level i, Algorithm 11 carefully implements this by first permuting the implicitly computed vectors $P\mathbf{a}$ in their new positions and then solving the linear systems for the new P vectors. Because of the tree structure, the total storage is l_{imax-1} which is always less than half than the final number of probing vectors at level l_{imax}—more accurately it would be less than $\min_j s(j, l_{imax-1})$.

Computationally, at level i, we have avoided the solution of l_{i-1} systems of equations at the expense of l_{i-1} inverse FFTs in (3.32), or a $\mathcal{O}(Nl_{i-1} \log l_{i-1})$ cost. Moreover, this is more elegant and less expensive than a brute-force Gram-Schmidt which costs $\mathcal{O}(Nl_{i-1}^2)$ at level i. Finally we remind the reader that this method works for hierarchical colorings on arbitrary graphs.

3.6 Performance Testing

We investigate the performance of our algorithm for lattices in two ways. First we show that the increased cost of the algorithmic extensions is not excessive. Second, we show that the probing vectors produced by our algorithm for lattices whose dimensions have sizes that are not powers of 2 provide better trace estimation than the vectors from the original algorithm in [66].

Our experimental results shown in Table 3.1 indicate that the increased computation that our method requires over the original algorithm is reasonable, given the short running times involved even for very large lattices. At the same time, the algorithm is still

Algorithm 11 GenerateAndPerformProbingVector_general(s, l, n, i_{max})

```
% Input: $s(j,i)$: number of colors the $j$-th block splits into at the $i$-th level,
%    $n_{(j,i,k)}$: the number of nodes in the color $k$ subgraph of block $j$
%    $l_{i-1}$: the number of colors at level $i-1$, also the number of blocks at level $i$
%    $i_{max}$: maximum desired level
% Output: The probing vectors $\mathcal{Z}$ at level $i_{max}$.
1: $\mathcal{Z} \leftarrow [\ ]$, $P_1 \leftarrow [\ ]$
2: $\mathcal{F}_{s(1,0)} \leftarrow \text{fft}(I_{s(1,0)})$
3: Build $\mathcal{Z}_{(1,0)}$ using (3.29) and the coloring permutation
4: $P \leftarrow [\ A^{-1}\mathcal{Z}_{(1,0)}]$
5: for $i = 1 \to i_{max}$ do                                   % Level $i$
6:     $P \leftarrow \text{ifft}(P^H)^T$
7:     $newpos(1) = 1$
8:     for $j = 2 \to l_{i-1}$ do                                % block $j$
9:         $newpos(j) = newpos(j-1) + s(j-1,i)$  % new positions of $P\mathbf{a}$ at level $i$
10:    end for
11:    $P(:, newpos) = P(:, 1 : l_{i-1})$                  % Permute to new positions
12:    for $j = 1 \to l_{i-1}$ do                                % block $j$
13:        $\mathcal{F}_{s(j,i)} \leftarrow \text{fft}(I_{s(j,i)})$
14:        Build $\mathcal{Z}_{(j,i)}$ using (3.29) and the coloring permutation
15:        for $k = 2 \to s(j,i)$ do                           % color $k$ in block $j$
16:            $P(:,k) \leftarrow A^{-1}\mathcal{Z}_{(j,i)}(:,k)]$
17:        end for
18:    end for
19: end for
```

	Original Method	Extended Method	
Lattice	**Time(ms)**	**Time(ms)**	**Time ratio**
8^4	12	39	3.3
16^4	187	673	3.5
32^4	3141	12156	3.8
4096^2	56228	277045	4.9
256^3	55435	266676	4.8
64^4	54598	252687	4.6

Table 3.1: Table showing run times of the new algorithm compared to the original. Results obtained on an Intel i7 860 clocked at 2.8 GHz.

embarrassingly parallel, since each point can be reordered independently. Given the low runtimes we obtain compared to the cost of solving the linear systems during probing, we have not investigated this option. Further, we observe that the dimensionality of the lattice does not impact the performance of the algorithm.

Finally, we examine the trace estimate produced on a model lattice problem, and compare it with the trace estimate produced by using a truncated permuted Hadamard matrix produced by the original hierarchical algorithm. This is essentially the same as applying an incorrect red-black coloring to the sublattices at each level, ignoring the links which are miscolored at the borders of each sublattice. While this will cancel the error from the most important parts of the structure of A^{-1}, it would still be leaving out important connections between sublattices. Therefore, we expect larger trace estimate errors, especially as the algorithm goes further down the coloring hierarchy. As we can see in Figure 3.14, the new algorithm does indeed perform significantly better, providing a much better trace estimate than the original method.

3.7 Conclusion

We have provided several extensions to the algorithm for hierarchically coloring and probing lattices. By formalizing the use of sublattices in the algorithm, we have made the algorithm easier to reason about. This allowed us to improve its flexibility, enabling the

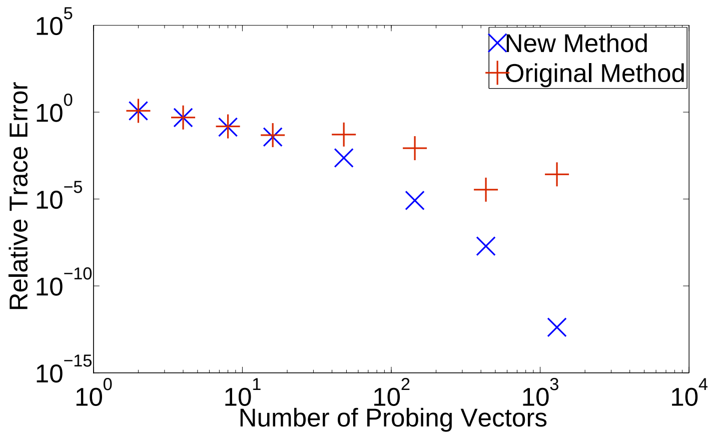


Figure 3.14: Comparison of the two methods on a 2D lattice with common factors $2 \times 2 \times 3 \times 3 \times 5$. For the common factors of two, the methods are the same, but once these are exhausted the improved method has much lower error.

algorithm to handle lattices with arbitrary dimensions, as long as the sizes of those dimensions share common prime factors. These improvements come at minimal computational cost and retain the ease of parallelization that was an attractive feature of the original algorithm. Finally, we have introduced a method of creating probing vectors, both for the case where the colorings split evenly into the same number of colors, and for the case where the coloring does not split evenly. We note that these methods of creating probing vectors can be applied to any matrix, not just those arising from lattices.

Chapter 4

Estimation of diag(f(A)) in the general case

In Chapter 3 we introduced methods to compute **Diag**$(f(A))$,in the case where the A in question arises from PDEs with certain geometric properties. Unfortunately, there are many interesting applications such as those discussed in 2.1.1 and 2.1.3, which give rise to matrices that do not have these useful properties.

This forces us to return to the original probing approaches for discovering structure in the matrix [62]. As discussed in 2.2.2 probing exploits the structure of $f(A)$ to achieve an accurate result more quickly than statistical methods, yet can be used on matrices with an unkown structure. The basis of the method is to take some polynomial approximation $q_n(A) \approx f(A)$. If the convergence of this polynomial approximation is fast this yields information about the most important elements of $f(A)$. The original approach involved using a Neumann series polynomial for A^{-1}, which implies that, if the diagonal elements of A are nonzero, the nonzero structure of $q_n(A)$ is the same as that of A^n. The graph coloring of A^n can then be used as an approximate coloring for the graph of A^{-1}, in the sense that the coloring constraints will only be violated by edges of small weights. This provides useful information about the structure of A^{-1}, which can be used as in Figure 2.3. Unfortunately, probing requires raising A to high powers to achieve more accurate results. This is not feasible for many matrices, since it can require significant amount of computation and perhaps even more importantly, a significant amount of storage.

In this chapter we propose methods based on the structural and spectral characteristics of $f(A)$ directly. These methods can be combined with the statistical methods of [22] to achieve the attractive features of these statistical approaches, as well as the benefits of probing.

4.1 Graph Coloring

Probing as introduced in Chapter 2.2.2 works by attempting to find the most important elements of A^{-1}, or equivalently, a sparsified version of A^{-1} matching the largest magnitude elements of A^{-1}. It does this by taking a polynomial approximation $q_n(A) \approx A^{-1}$. In the original proposal of probing, the polynomial was chosen as the Neumann series. If the polynomial converges, then its graph approximates a sparsified structure of A^{-1} where the most important (large in magnitude) elements are kept. The goal is to find a degree for which the polynomial graph is sufficiently sparse to be colored with a reasonable number of colors. This is because if the graph associated with a matrix has a k-coloring, then the diagonal of A can be recovered with k matrix vector multiplications. To see how this recovery is possible, consider the structure of the matrix if all the nodes that share a color are reordered to be adjacent. Since sharing a color implies a lack of communication between the nodes, the matrix can be permuted in a block diagonal form, as shown in Figure 2.3. Then, if we create the p^m probing vector as in (2.4) then Ap^m holds the diagonal elements in the positions of the m-th block.

Since coloring algorithms only consider if two nodes are connected or not, the weights of the edges are not used. Thus instead of the entire Neumann series $q_n(A)$, only A^n need be computed since the structure of the two matrices are the same. Since A^n shows all the connections between nodes after n hops, we can either find a distance 1 coloring of A^n, or a distance n coloring of A. Both operations have similar complexity. Once the probing vectors p^m have been obtained, we use an iterative method to solve $Ax = p^m$. Since iterative methods applied for large matrices may be expensive and slow to converge to the desired accuracy, we need to minimize the number of colors used in order to keep the number of probing vectors produced low.

Problems arise when raising A^n to a high enough power for $\|q_n(A) - A^{-1}\|$ to be small. Even if A is sparse, A^n can become dense quickly which increases both the computational and the storage requirements of the method. Additionally, while probing is intended to exploit the structure of A, it throws out interesting structural information by not considering the strength of the connections. Finally, the colors produced by probing at different levels are unlikely to be nested subsets of each other. Nodes which previously were given separate colors may later on be assigned the same color, which means that results with previously generated probing vectors cannot be reused if increased accuracy is desired; the entire computation must be repeated from scratch.

In this chapter we investigate two types of methods for providing better colorings than probing. The first set of methods are similar to probing in that they are structural; they try to directly detect and exploit the structure of $f(A)$.

The second type of method attempts to use computed spectral information of A to find an appropriate coloring for $f(A)$—similar to the well known techniques of spectral clustering. While the algorithm we have developed is not very efficient, it serves as an interesting way to examine the issues involved with the design of spectral methods, and a starting point for future research into these types of approaches. It also provides an upper bound on how well such methods could work.

We combine both types of methods with the statistical methods of [22] discussed in Chapter 2.2.1, observing in many experiments that we can obtain a more accurate error bound for $\mathbf{Diag}(f(A))$ than would be possible using only statistical methods.

4.2 Statistical Considerations

While both [62] and [27] consider probing and probing-like ideas from the standpoint of a deterministic process, in practice our experiments show error from the statistical methods in [22] being much smaller than the deterministic error reported by probing. Additionally, many applications (such as QCD) require information about the error and require the error to be unbiased. Fortunately it is possible to combine statistical methods with probing as shown in [66]. By generating a random vector ζ^m for each color block m and performing elementwise multiplication with each probing vector to form $p^m \odot \zeta^m$, we make our estimator statistically unbiased. In addition, if probing correctly identifies the smallest elements of A^{-1} and groups them into colors, then the variance and thus the error estimate will be reduced, while providing the added benefit of a statistical confidence interval for the results.

As noted in Table 2.1 the variance of Hutchinsons method is $2(\|A\|_F^2 - \sum_{i=1}^N A_{ii}^2)$. If a set of s random vectors ζ is choosen according to [22], and ζ_j^k refers to the j-th component of the k-th sample, then the diagonal estimator is given in [27] as

$$\text{TdiagA}_i = A_{ii} + \sum_{j=1, j \neq i}^{N} a_{ij} \frac{\sum_{k=1}^s \zeta_i^k \zeta_j^k}{\sum_{k=1}^s (\zeta_i^k)^2}. \tag{4.1}$$

If the ζ are i.i.d. drawn from the Rademacher distribution, then for $j \neq i$ the expectation $E[\sum_{k=1}^s \zeta_i^k \zeta_j^k] = \sum_{k=1}^s E[\zeta_i^k \zeta_j^k] = 0$, implying that $E(\text{TdiagA}_i) = A_{ii}$. We are interested in computing the variance $E[\text{TdiagA}_i^2] - E[\text{TdiagA}_i]^2 = E[\text{TdiagA}_i^2] - A_{ii}^2$. For this, we notice that $E[(\sum_{k=1}^s \zeta_i^k \zeta_j^k)(\sum_{l=1}^s \zeta_i^l \zeta_m^l)] = s$ only when $k = l$ and $j = m \neq i$. Since

$\sum_{k=1}^{s}(\zeta_i^k)^2 = s$, we have

$$\begin{aligned} E[\mathrm{TdiagA}_i^2] &= a_{ii}^2 + 2a_{ii}\sum_{j=1,j\neq i}^{N} a_{ij} E\left[\frac{\sum_{k=1}^{s}\zeta_i^k\zeta_j^k}{\sum_{k=1}^{s}(\zeta_i^k)^2}\right] \\ &\quad + \sum_{m,j=1,\ m,j\neq i}^{N} a_{ij}a_{im} E\left[\frac{\sum_{k=1}^{s}\zeta_i^k\zeta_j^k}{\sum_{k=1}^{s}(\zeta_i^k)^2}\frac{\sum_{l=1}^{s}\zeta_i^l\zeta_j^l}{\sum_{l=1}^{s}(\zeta_i^l)^2}\right] \\ &= a_{ii}^2 + \sum_{j=1,j\neq i}^{N} a_{ij}^2. \end{aligned} \tag{4.2}$$

Thus, $\mathrm{Var}(\mathrm{TdiagA}_i) = \sum_{j=1,j\neq i}^{N} a_{ij}^2$, which means the estimator of each element depends on the off-diagonal elements of its row.

We can now bound how large the standard deviation is relative to the trace, which provides information on the number of digits that can be achieved using these estimators. For symmetric positive definite matrices and a given number of samples s, we have

$$\frac{\frac{\sqrt{2(\|A\|_F^2-\sum A_{ii}^2)}}{\sqrt{s}}}{\mathbf{Tr}(A)} \leq \frac{\frac{\sqrt{2\|A\|_F^2}}{\sqrt{s}}}{\mathbf{Tr}(A)} = \frac{\frac{\sqrt{2\mathbf{Tr}(A^2)}}{\sqrt{s}}}{\mathbf{Tr}(A)} = \frac{\frac{\sqrt{2\sum\lambda_i^2}}{\sqrt{s}}}{\sum\lambda_i} \leq \frac{\frac{\sqrt{2(\sum\lambda_i)^2}}{\sqrt{s}}}{\sum\lambda_i} = \frac{\sqrt{2}}{\sqrt{s}}. \tag{4.3}$$

Although the bound may be pessimistic, it indicates that we should not have problems obtaining good relative estimates. On the other hand, for the diagonal estimator,

$$\frac{\sqrt{\sum_{j=1,j\neq i}^{N} a_{ij}^2}}{\sqrt{s}|a_{ii}|} \leq \frac{\|A(i,:)\|}{\sqrt{s}|a_{ii}|} \leq \frac{\|A\|}{\sqrt{s}|a_{ii}|}, \tag{4.4}$$

so the relative error can be quite large for some diagonal entries. One exception, which is often useful in practice, is the case of diagonally dominant matrices. Then,

$$\frac{\sqrt{\sum_{j=1,j\neq i}^{N} a_{ij}^2}}{\sqrt{s}|a_{ii}|} \leq \frac{\sum_{j=1,j\neq i}^{N}|a_{ij}|}{\sqrt{s}|a_{ii}|} \leq \frac{1}{\sqrt{s}}, \tag{4.5}$$

and the stochastic estimator can provide good relative estimates.

It is possible to investigate how well probing performs compared to statistical methods such as Hutchinson's by modeling the magnitude of the elements removed by probing. As noted in Table 2.1 the variance of Hutchinson's method is $2(\|A\|_F^2 - \sum_{i=1}^n A_{ii}^2)$. Since in the Hutchinson method the diagonals do not contribute to the variance, in our analysis we need only consider what happens to the off-diagonal part of the matrix. Because we will refer to this portion of the matrix frequently, we define $\tilde{A}$ as the matrix A with the diagonal removed.

When using Hutchinson's method, the variance comes from the entire $\tilde{A}$ matrix. In contrast, when using probing with a statistical estimator as above, the variance comes only from inside the block diagonals, such as those seen in Figure 2.3. This is because the contributions from outside the block diagonals are zeroed out. Since the block diagonals are the only parts of the matrix contributing to the variance estimation, we can simply compute $2\|\tilde{A}_i\|_F^2$ for each block, and then sum the results for all k blocks. This is equivalent to considering k independent Hutchinson methods, one for each color with variance $2\|\tilde{A}_i\|_F^2$.

Suppose that the k color blocks are all of equal size. Then, if N is the dimension of A, there are $k\frac{N^2}{k^2} - N = \frac{N^2}{k} - N$ elements of $\tilde{A}$ contributing to the variance, i.e., only those in block diagonal. Equivalently, the k-coloring discards $\frac{k-1}{k}N^2$ elements from contributing to the variance. Assume in addition that we have sorted all the elements of $\tilde{A}$ in monotonically decreasing order and that increasing the number of colors discards off-diagonal elements starting from the largest in the list. More specifically, $G = \text{sort}(\tilde{A}_{ij}^2)$ would be an array of N^2 elements with the zero diagonal elements at the end. We model this array as a monotonically decreasing function $g(x)$, where the input $x = \frac{k-1}{k} \in [0, 1]$ is the percentage of the discarded elements.

We can begin to model the variance that is removed given certain assumptions about $g(x)$. First, we assume that probing finds the largest elements of $\tilde{A}$ in monotonically decreasing manner and removes them. Then, with 2 probing vectors we have $Var_2 = \int_{\frac{1}{2}}^1 g(x)dx$, since with two probing vectors half the elements will be discarded. In general, with k probing vectors (i.e., with k colors), $Var_k = \int_{1-\frac{1}{k}}^1 g(x)dx$. If $g(x)$ is constant, i.e., all the elements of A are the same, then we have $Var_k/\|\tilde{A}\|_F^2 \sim O(\frac{1}{k})$. On the other hand, if $g(x) \sim (1-x)$, then we have that $V_k/\|\tilde{A}\|_F^2 \sim O(\frac{1}{k^2})$.

This analysis implies that even if there is no structure to be exploited, as long as the nodes are divided up into color blocks of equal size, probing will do at least as well as the purely statistical methods of Hutchinson. Therefore algorithms should attempt to make the color sizes as equal as possible. Moreover, as long as significant structure exists in the matrix, probing should be able to outperform statistical methods.

4.3 Structural Methods

As previously discussed, the major problem with probing arises when the series $q_n(A)$ converges slowly to $f(A)$. To see this issue in action consider the example in Figure 4.1. In this example, we consider a 4D Laplacian with periodic boundary conditions and compare probing using polynomials of order ranging from 1–8 against computing the pseudo-inverse $A^\dagger$ directly, dropping the elements of the smallest magnitude (that is, sparsifying the matrix), and then coloring the resultant matrix. We take the pseudo-inverse, because this operator has one zero valued eigenvalue. This coloring of the sparsified version of $A^\dagger$, while not practical provides information on how close to the optimal coloring the coloring provided by probing is. As we can see, probing is close to the optimal when few colors are considered, but as the number of colors increases the difference between the two methods becomes significant.

To address this, we seek to capture the structure of $f(A)$ more directly. We propose here two methods to achieve this. The first method attempts to capture the off-diagonals of $f(A)$. Many matrices of interest have an inverse with a banded structure. This implies that we should be able to find a coloring that captures most of the structure of $f(A)$. Indeed, this was behind the original idea of probing, where the authors note that if the coloring distance is increased far enough, then the major off-diagonals of the matrix will be captured. Similarly, the authors of [27] note that if a large enough number of columns of the

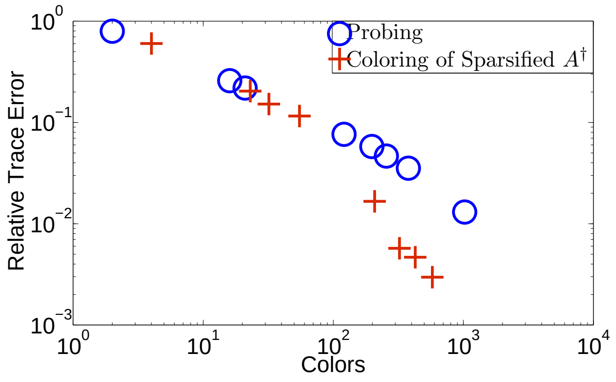


Figure 4.1: Probing vs Coloring the structure of $L^\dagger$ directly, where the percentage of the weight of $L^\dagger$ retained varies from .1 to .5. in .05 increments. As the number of colors increases probing struggles to capture the structure of $L^\dagger$.

Hadamard matrix is used, the off-diagonals of interest will be removed from contributing to the error estimate. However, these are both indirect methods of achieving the goal of discovering which nodes of the underlying graph of $f(A)$ are connected via an off-diagonal with large values. To gain insight into how it might be possible to discover such connections, consider the sparsity plot of the matrix containing several off-diagonals below. If we take a sampling of the columns of the matrix, and we assume that the line the off-diagonals follow has a slope of 1, if any two columns were sampled at a distance k to each other, then if we shift the samples up by k, the non-zero structure of the samples will be the same, as can be seen in Figure 4.2, where we sample four columns.

Of course, in the case of $f(A)$, the output is likely to be dense, so matching up samples of $f(A)$ and seeing where the non-zeros match is not possible. However, if we obtain some columns of $f(A)$ using an iterative method, and then sparsify them by dropping some percentage of the elements with the smallest absolute values, we can then compare the structure of the samples, and check to see where the non-zero values line up. If a significant number of the sparsified samples share the same non-zero structure, it is likely there is an off-diagonal at that location. We can then use these off-diagonals to compute an approximate coloring for $f(A)$. This approach is shown in Algorithm 12.

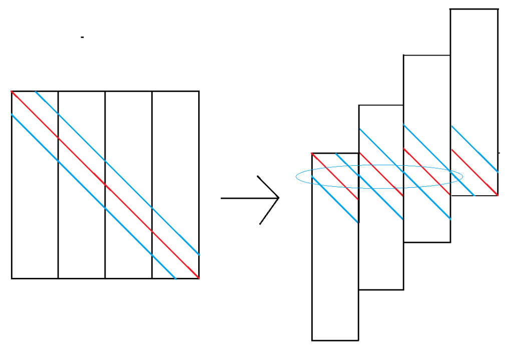


Figure 4.2: Sampling 4 columns from A and then shifting to detect if they share an off-diagonal

The second method we consider is based on the matrix-matrix multiplication approximation method proposed in [2]. They note that if some samples of a column of matrix $v = A(:, i)$ are obtained, then $vv^T \approx A^2$. Since we have obtained the columns v of $f(A)$ to estimate the off-diagonals of $f(A)$, we have $vv^T \approx f(A)^2$. Since we want $ww^T = f(A)$, we preform a QR factorization of v. Then $v = QR$ and $(vv^T)^{1/2} = (QRR^TQ^T)^{1/2} = Q(RR^T)^{1/2}Q^T$. We can obtain $(RR^T)^{1/2}$ using SVD, since R is a small dense matrix. This procedure is shown in Algorithm 13.

The approximation ww^T, however, is still dense. To ensure a sparse approximation to $f(A)$, we again sparsify w by dropping the smallest magnitude elements. While it is likely that $\|ww^T - f(A)\|$ will be large, what is important for our application is not the difference in the value of the elements of the two matrices, but how close the orderings of the elements of the matrices are, since these are the elements that probing will discard. To measure this, consider the two sets o_1 and o_2 that each contains the index pairs (i, j) of the non-zero elements of the matrix $f(A)$ and ww^T, respectively. Define p_1 to be the permutation of the o_1 that sorts the corresponding elements of $f(A)$ from largest to smallest magnitude. Define also p_2 the permutation of o_2 that sorts the elements of ww^T from largest to smallest magnitude. Since ww^T is a sparse matrix, it has fewer elements, $z = \text{size}(o_2) \leq \text{size}(o_1)$. We then want to check how closely the index pairs (not the matrix

Algorithm 12 $[D]$ =DetectOffdiags(v, ϵ)
% Input: v columns of $f(A)$, L array containing column locations, a tolerance ϵ
% Output: D a matrix of important off-diagonals

```
 1: v ← Sparsify(v)
 2: newv ← []
 3: for $i = 1 \rightarrow$ size(v,2)  do
        newv ← [newv shift(v(:,$i$),-L($i$))]
 4: end for
 5: newvsums ← sum(abs(newv));
 6: diaglocations ← [];
 7: for $i = 1 \rightarrow$ size(v,1)  do
 8:     if newsums($i$) $\geq \epsilon$  then
        diaglocations($i$) ← 1;
 9:     end if
10: end for
11: D ← diag(diaglocations)
12: return D
```

Algorithm 13 $[w]$ =CreateW(v, ϵ)
% Input: v columns of $f(A)$
% Output: w for use in approximation ww^T

```
1: $v \leftarrow$ sparsify$(v, \epsilon)$
2: $[Q,R] \leftarrow$ QR$(v)$
3: $[U,S,V] \leftarrow$ SVD$(v)$
4: $w \leftarrow QUS^{1/2}V^T$
5: return $w$
```

elements) in $o_2(p_2)$ match the first z index pairs in $o_1(p_1)$ by computing $o_1(p_1) \cap o_2(p_2)$. If the size of the intersection is equal to z, then we have the best possible approximation over all matrices with that number of non-zero elements, in the sense that we have captured the z most important elements of $f(A)$.

It is interesting to contrast the parts of the structure of $f(A)$ that the two methods find. Algorithm 12 attempts to detect the global structure of the matrix, while Algorithm 13 detects more local structure. Figure 4.3 shows the effect of progressively increasing the number of columns from $f(A)$. As the number of columns increase, the local areas of the matrix $f(A)$ that are well approximated expand. Our experiments reveal that there is a transition point with this method, where after a certain number of columns, the coloring produced starts to be extremely effective, although for large matrices this point may come too late to be practical. A similar transition point can be seen with the density, where as the sparsity of v decreases, there is a point where the coloring becomes substantially better, although again, this point can come too late to be practical.

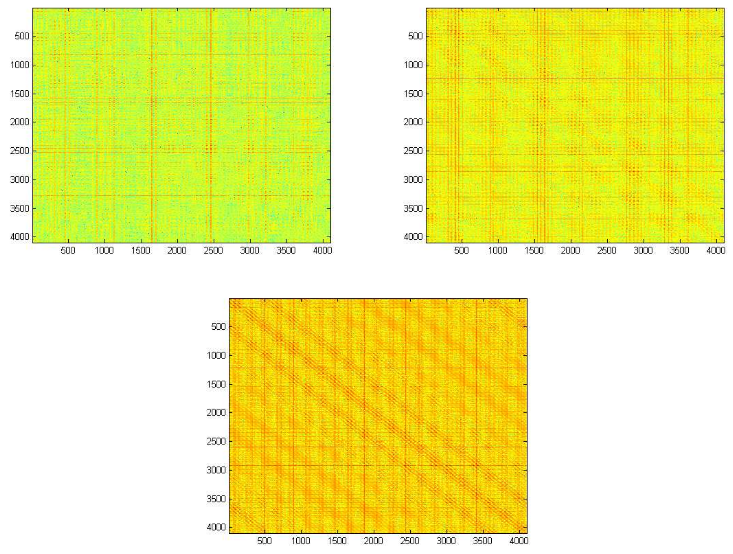


Figure 4.3: Approximation of the pseudo-inverse of a Laplacian with periodic boundary conditions with 10, 100, and 1000 vectors in the vv^T approximation. Here the vectors v are not sparsified, the figure shows how in the best case of an unsparsified v, vv^T contains mostly local structure, until a significant number of vectors v are supplied.

A significant drawback of these two methods is that it is difficult to provide any bounds on the error associated with using the colorings they produce. Matrices which have a significant number of off-diagonals or other highly ordered structure will likely have this structure captured by these methods. Less structured matrices may be more challenging, and it may be hard to tell the difference a-priori. Another drawback is that it is difficult to tell what level of sparsification should be applied to the vectors used by these methods. One possible method is for the user to supply a maximum number of colors representing the limit of the number systems $f(A)z$ to be solved, and then preform bisection on the densities of the two methods, doubling or halving the value until the required number of colors are obtained. However, this still leaves open the question of the starting densities to be used. While for some classes of matrices the user may have a good idea what will work best, in the general case it is not possible to know the optimal value. A possible heuristic is to examine the sampled columns v, and determine where the sharpest drop is in the value of the sorted components. This point can then be used as a starting point which can be refined by the previously described bisection method.

While we leave further investigation for future work, we note the similarities between this method and the Nyström method [3]for approximating symmetric semi positive definite matrices. In the classic Nyström method a certain number of columns C are randomly selected from the matrix A for which an approximation is desired. The intersection matrix W is then formed by finding the intersection of the rows and columns of C, or $C = A(:, cols); W = C(cols, :)$; in matlab notation. The pseudo inverse W^+ is then formed. If many columns of C are selected, it may not be feasible to find the pseudoinverse of W, in which case the best rank k approximation $W_k^\dagger$ is formed. Then $A \approx CW^\dagger C^T$. There are several variations on this method which are possible, such as taking an ensemble of such approximations or replacing the intersection matrix W with the matrix $C^\dagger A(C^\dagger)^T$. The question then is how to best select C. If additional columns of C could be selected adaptively this method could allow the derivation of error bounds, in contrast to our current method for which it is difficult to prove any bounds. If the bounds are tight enough, it might even make sense to directly compute $\mathbf{Tr}(CW^{\dagger}C^{T})$ or $\mathbf{Diag}(CW^{\dagger}C^{T})$, and dispense with coloring entirely, relying on the theoretical error bounds instead of the statistical error measurements for guarantees for the accuracy.

4.4 Spectral Methods

Spectral methods for finding communities in a graph have been well understood for some time [6]. The general basis for these methods comes from the graph partitioning problem. Given a graph $G = (V, E)$ and associated graph Laplacian L,

$$L_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}, \tag{4.6}$$

the nodes must be divided into groups of nearly equal size, S and its complement $\bar{S}$, in such a way that the weight of the edges between the groups is minimized. We can formalize this problem as follows. Define $\phi(S)$ as

$$\phi(S) = \frac{|E(S, \bar{S})|}{\min(|S|, |\bar{S}|)} = \frac{\sum_{(i,j)\in E}(x_i - x_j)^2}{\sum_{i<j}(x_i - x_j)^2}. \tag{4.7}$$

Then define the isoperimetric number of a graph as the value of the minimum cut

$$\phi_{opt} = \min_{S\subseteq V} \phi(S). \tag{4.8}$$

Unfortunately, this problem is NP-Complete. However, if instead of requiring the elements of the solution vector x to be in $\{-1, 1\}$, the problem is relaxed by allowing the solution elements to take real values, then we can obtain a solution as follows

$$\phi_{opt} \approx \min_{x\in\mathbb{R}^n} \frac{\sum_{(i,j)\in E}(x_i - x_j)^2}{\sum_{i<j}(x_i - x_j)^2} = \min_{\substack{x\in\mathbb{R}^n \\ x\perp\mathbf{1}}} \frac{\sum_{(i,j)\in E}(x_i - x_j)^2}{n\sum_{i=1}^{n}(x_i)^2} = \min_{\substack{x\in\mathbb{R}^n \\ x\perp\mathbf{1}}} \frac{x^T A x}{n x^T x}. \tag{4.9}$$

By the Courant-Fischer Theorem, this is is minimized by v_2, the eigenvector of the second smallest eigenvalue l_2 of L, known as the Fiedler vector.

While this relaxation is sufficent for regular graphs, frequently the normalized edge cut is desired for irregular graphs. If we define $vol(S) = \sum_{v_i \in S} deg(v_i)$, then the normalized edge cut is defined as

$$\hat{\phi} = \frac{|E(S, \bar{S})|}{min(vol(S), vol(\bar{S}))}. \tag{4.10}$$

In contrast to (4.7), (4.10) divides by the degree of the nodes in the volume instead of simply the number of vertices. This generally turns out to be a more robust measure [6]. In this case, the solution of the relaxed problem is related to the spectrum of the normalized graph Lapalcian, $\mathcal{L}$, defined as in (4.11), where D is the diagonal matrix of the degrees of the nodes of the graph, and L is the graph Laplacian of (4.6),

$$\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}. \tag{4.11}$$

By performing the same relaxation as (4.9) we achieve the Cheeger bounds on $\hat{\phi}_{opt}$, where λ_2 is the second smallest eigenvalue of $\mathcal{L}$,

$$\frac{\hat{\phi}_{opt}^2}{2} \leq \lambda_2 \leq 2\hat{\phi}_{opt}. \tag{4.12}$$

It is possible to continue this process of finding an approximation to the best edge cut recursively by computing the Fiedler vector of the induced subgraph of each partition. This allows for an arbitrary number of partitions of the graph to be obtained. This process is known as recursive spectral bisection [7].

While recursive bisection is the simplest method of spectral clustering, there are many other proposals. In fact, [8] lists five distinct classes of algorithms for finding k-clusters, based on the lower part of the spectrum of $\mathcal{L}$. The first of these he describes as linear ordering, where the nodes are reordered based on the smallest eigenvector recursively, i.e., recursive spectral bisection. We investigate a heuristic based on this idea, and leave a more through investigation of ways to adapt algorithms for spectral clustering to our problem of interest for future work.

In seeking to exploit the structure of a graph for trace estimation, probing tries to solve the opposite problem to that of community detection. Instead of trying to find communities with a large amount of intra-community interactions, probing seeks to find groups of nodes that have no or few interactions between each other. In this case, [9] has shown that the function we should seek to optimize is

$$\bar{\phi}(S) = \frac{2|E(S,\bar{S})|}{vol(S) \cup vol(\bar{S})}, \tag{4.13}$$

a measure analogous to the normalized cut in (4.10). As in the community detection problem, we seek to optimize this function as

$$\bar{\phi}_{opt} = \max_{S} \frac{2|E(S,\bar{S})|}{vol(S) \cup vol(\bar{S})}. \tag{4.14}$$

While this is again an intractable problem, the relaxed version again admits a solution which is related to the spectrum of $\mathcal{L}$, with the solution vector being the largest eigenvector of the graph Laplacian. It is possible to bound the error of the relaxed version of this problem using an adjusted version of the Cheeger bounds [9]

$$\frac{1-\bar{\phi}_{opt}^2}{2} \leq 2 - \lambda_N \leq 2(1-\bar{\phi}_{opt}). \tag{4.15}$$

The relationship between (4.14) and the relaxed quadratic form is more difficult to see in this case than it is for (4.10). The derivation of the upper and lower bounds in (4.15) is rather involved and can be found in [9]. Here we present the derivation only of the upper bound. In [9] the authors partition the graph nodes into three sets; V_1, V_2 are two bipartite sets, and $V_3 = \overline{V_1 \cap V_2}$ are the rest of the nodes that cannot be split in bipartite sets. Then they note $vol(V_i) = \sum_{j=1}^{3} |E(V_i, V_j)|$. With this, (4.14) can be rewritten as

$$\bar{\phi}_{opt} = \max_{V_1,V_2} \frac{2|E(V_1,\bar{V_1})|}{\sum_{j=1}^{3}|E(V_1,V_j)| + \sum_{j=1}^{3}|E(V_2,V_j)|}. \tag{4.16}$$

Hence, the connection to a quadratic form based on the eigenvectors of $\mathcal{L}$ follows,

$$\begin{aligned}
\lambda_N &= \max_{\substack{x\in\{\mathbb{R}\}^n \\ x\perp\mathbf{1}}} \frac{x^TAx}{x^Tx} \\
&\geq \frac{(\frac{1}{vol(V_1)}+\frac{1}{vol(V_2)})^2|(E(V_1,V_2)| + (\frac{1}{vol(V_1)})^2|E(V_1,V_3)| + (\frac{1}{vol(V_2)})^2|E(V_2,V_3)|}{\frac{1}{vol(V_1)}+\frac{1}{vol(V_2)}} \\
&\geq \frac{(vol(V_1)+vol(V_2))^2}{2vol(V_1)vol(V_2)}\frac{2|E(V_1,V_2)|}{vol(V_1)+vol(V_2)} + \frac{min(vol(V_1),vol(V_2))}{max(vol(V_1),vol(V_2))}\frac{|E(V_1\cup V_2,V_3)}{(vol(V_1)+vol(V_2)} \\
&\geq 2\bar{\phi}_{opt} + \frac{min(vol(V_1,vol(V_2))}{max(vol(V_1),vol(V_2))}\frac{|E(V_1\cup V_2,V_3)|}{(vol(V_1)+vol(V_2))} \\
&\geq 2\bar{\phi}_{opt}.
\end{aligned}$$

4.4.1 Spectral k-partitioning for the matrix inverse

Since our goal is to find the structure of the matrix inverse, we turn our attention to $\mathcal{L}^\dagger$, instead of $\mathcal{L}$. We need the pseudoinverse because the graph Laplacian is singular, $\mathcal{L}D^{\frac{1}{2}}\mathbf{1} = \mathbf{0}$. Note that $\mathcal{L}^\dagger D^{\frac{1}{2}}\mathbf{1} = \mathbf{0}$, and $\mathcal{L}^\dagger$ has the same eigenvectors as $\mathcal{L}$ only ordered in the opposite order. Let us consider $\mathcal{L}^\dagger$ as a weighted graph with adjacency matrix, $\Gamma = \mathcal{L}^\dagger - \mathbf{Diag}(\mathcal{L}^\dagger)$, and ignore the fact that some weights may be negative. Then for the (weighted) Laplacian of Γ,

$$\begin{aligned}
L_\Gamma &= \mathbf{Diag}(\Gamma\mathbf{1}) - \Gamma = \mathbf{Diag}(\mathcal{L}^\dagger\mathbf{1} - \mathbf{Diag}(\mathcal{L}^\dagger)\mathbf{1}) - \mathcal{L}^\dagger + \mathbf{Diag}(\mathcal{L}^\dagger) \\
&= -\mathbf{Diag}(\mathcal{L}^\dagger) - \mathcal{L}^\dagger + \mathbf{Diag}(\mathcal{L}^\dagger) = -\mathcal{L}^\dagger. \qquad (4.17)
\end{aligned}$$

This implies that the graph Laplacian of the pseudoinverse shares the same eigenvectors and in the same order as $\mathcal{L}$. Thus, the same partition is a solution to the relaxed version of (4.16) for both matrices.

This relation suggests a connection between spectral methods and probing. Probing considers the powers of A^n in order to learn about the structure of $f(A)$. Similarly, the power method takes matrix vector products with A on a starting vector v_0, computing $A^n v_0$ which is known to converge to the direction of the largest eigenvector. Thus, spectral method short cut the intermediate steps of the low order polynomial representations, and skip straight to the same information that the highest order polynomials provide.

The idea of finding communities with a low number of intra-community links using the top eigenvectors suggests that many of the algorithmic ideas used for spectral clustering might be adapted to the opposite problem. An obvious first algorithm is to obtain the largest eigenvector of the graph, and use it to divide the graph into two groups that are as close to bipartite as possible. Then we apply the method recursively on the induced subgraph for each group. We continue until a maximum number of groups is reached, or until the largest eigenvalue of the Laplacian of the partitioned matrices is too small to allow for a good bipartitioning. This method has the obvious advantage that the colors we produce will be nested subsets of each other, allowing us to continue probing without discarding previous results.

Unfortunately, this method has limited practicality when applied to $f(A) = \mathcal{L}^{\dagger}$. While the eigenvectors needed at the first level are easily obtainable since they match those of $\mathcal{L}$, the required eigenvectors at the next level are more difficult to compute. The recursive process requires the eigenvectors of submatrices of $\mathcal{L}^{\dagger}$, but we do not explictly have these submatrices available. We could use iterative methods to compute their eigenvectors, but each iteration would need a matrix-vector product with the submatrix we do not have. Therefore, for each matrix-vector product and for each submatrix, we must solve a linear system with $\mathcal{L}$. With even a few color blocks, this may cause the process to become infeasible. Further, the diagonal elements of the submatrices do not quite correspond to the elements of the graph Laplacians of the subgraphs of $\mathcal{L}^{\dagger}$, meaning we will obtain partitions which are not quite correct.

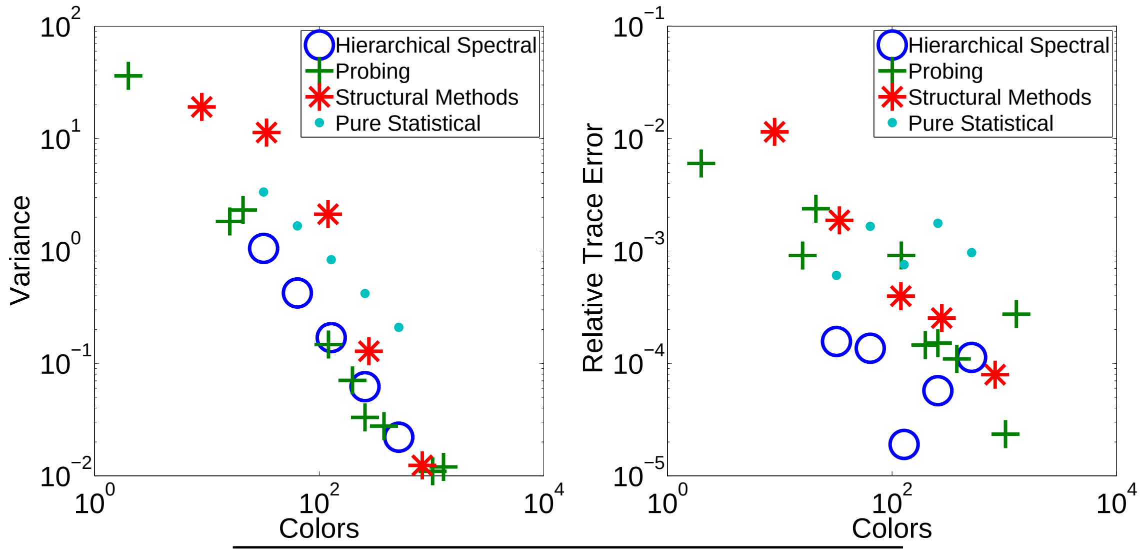
Nevertheless, we report this algorithm for the following reasons. First, if the discrepancy between the diagonal elements of $\mathcal{L}^{\dagger}$ and the appropriate Laplacian is not large, which is the case in our observations, then the algorithm has the same theoretical support as recursive spectral bisection for partitioning. In fact, for a given number of colors, the algorithm finds very good quality colorings, so it serves as a proxy for an upper bound for what spectral methods can achieve. Second, while the approach is not practical for $f(A) = \mathcal{L}^\dagger$, it may be practical for other $f(A)$, such as $f(A) = A^n$. We show our approach in Algorithm 14.

Algorithm 14 c ←SpectralBisection($\mathcal{L}$, k)
% Input: Laplacian matrix $\mathcal{L}$, desired number of partitions k
% Output: a coloring c

```
 1: [evecs, evals] ← eigs($\mathcal{L}$)
 2: n ← size($\mathcal{L}$,1)
 3: % Get a permutation for the sorted eigenvector v
 4: [vs, p] = ← sort(evecs)
 5: ip(p) ← [1:n]    % Inverse permutation
 6: m = floor(n/2)
 7: if $k == 0$ then
 8:     c(p(1 : m)) ← 1
 9:     c(p(m+1 : n)) ← 2
10: else
11:     $\mathcal{L}_1$ ← $\mathcal{L}$(1:m, 1:m)
12:     $\mathcal{L}_2$ ← $\mathcal{L}$(m+1:n, m+1:n)
13:     smallc$_1$ ← SpectralBisection($\mathcal{L}_1, k-1$)
14:     smallc$_2$ ← SpectralBisection($\mathcal{L}_2, k-1$)
15:     nextC ← [smallc$_1$ , smallc$_2$+max(smallc$_1$) ]
16:     % Return coloring to original ordering
17:     c ← nextC(ip);
18: end if
19: return c
```

4.5 Experimental Results

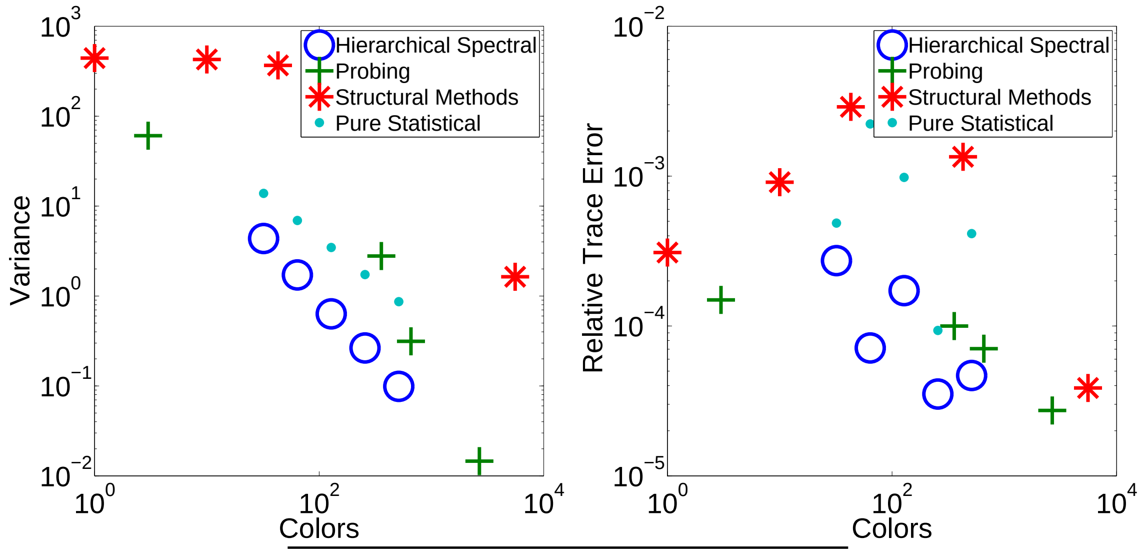
In this section we present two sets of experimental results. The first set are Laplacian matrices from various graphs, chosen to help test our spectral approach. For these these matrices, we form the peusdoinverse $\mathcal{L}^\dagger$, since the matrices are singular, and then use this inverse to directly compute the variance and the error. The second set are various matrices selected from the Matrix Market sparse matrix collection [12], chosen because they were used as tests cases for [27]. Since these matrices were all chosen to be invertible we form A^{-1} directly, and then use this inverse to compute the variance and error.



Density Range	Vs	Diagonals	Probing Powers
.1 − .5	50	20	9

Figure 4.4: The variance and relative trace error of the trace estimator for the 8^4 Laplacian with periodic boundary conditions.

While only the graph Laplacians have results for our spectral method, we show the results of our structural approaches for all the graphs. For each matrix we take 50 random columns v, and then apply the sparsity listed in the table below each graph. The number of columns in v is held constant, but the density is varied. We select a starting percentage of sparsification using the heuristics we described in Section 4.3. We then increase this density every step. While we initially increase the density by increments of 10 percent, if this proves to be either too little or too much change we use recursive bisection to adjust the increment. This process continues until the maximum color budget is reached. This maximum budget varies from matrix to matrix, because some of the examples are small, but is always set to be less then one thousand, which is the maximum we expect users to employ in real world scenarios. We also use our heuristics to pick the number of diagonals the model uses, which is then held constant thoughout the process. For the first set of

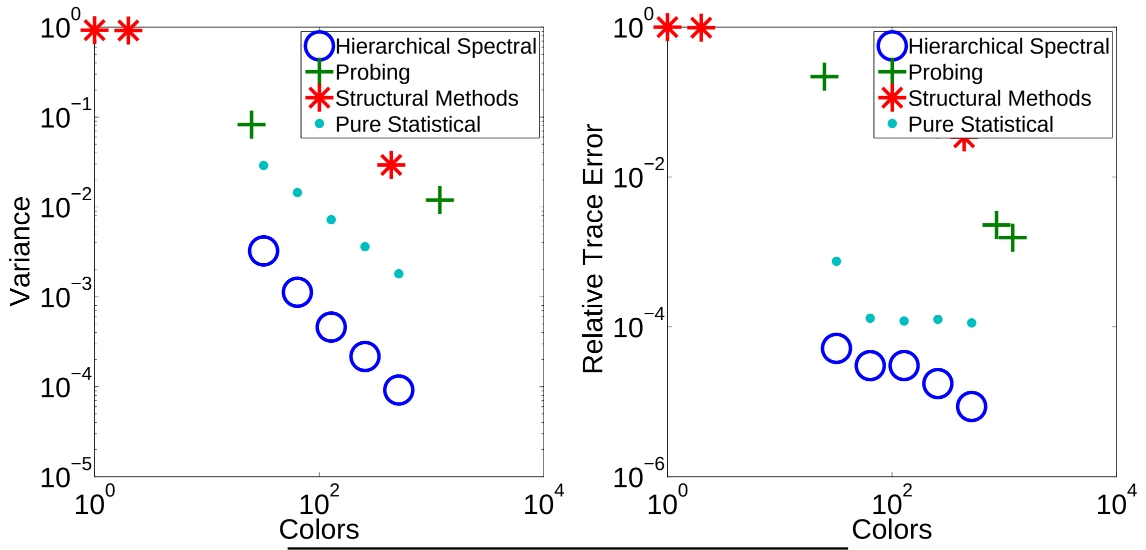


Density Range	Vs	Probing Powers	
.1 − .47	50	8	4

Figure 4.5: The variance and relative trace error of the trace estimator for a randomly generated scale free graph.

experiments we show only the statistical approach described in Section 4.2, forming new probing vectors as $\zeta \odot P$, where ζ is a noise vector. For the second set of results we show in addition the results with the original Hadamard probing vectors without the statistical approach, since these matrices were originally chosen to show how Hadamard vectors can remove specific diagonals. Finally, we also show how much the Hadamard method reduces the variance, which indicates how well it would preform if used as the starting point of a Monte Carlo method.

The first graph we examine in Figure 4.4 is the $4D$ Laplacian differential operator with periodic boundary conditions on a regular lattice. These types of graphs occur in LQCD. This is the graph Laplacian test case where the structural methods and classical probing perform the best, remaining competitive with the spectral method. This is most likely due to the highly structured nature of the lattice, where certain off-diagonals take up most of the weight of A^{-1}. Next we examine a synthetic scale-free graph with 10000 nodes, generated using the CONTEST MATLAB toolbox [13], the results of which we

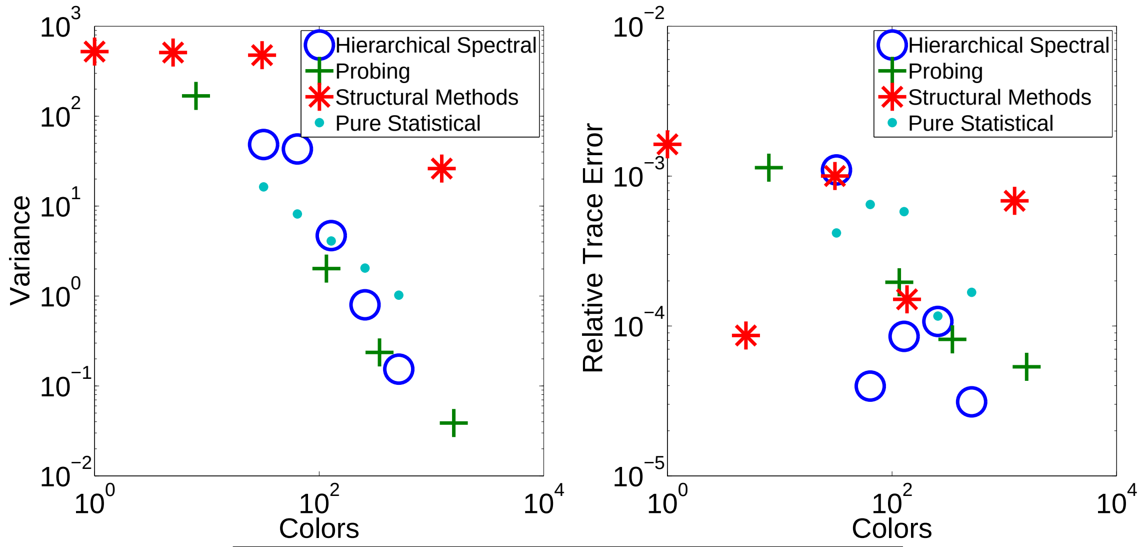


Density Range	Vs	Probing Powers	
$.1 - .68$	50	1	3

Figure 4.6: The variance and relative trace error of the trace estimator for the wiki-vote graph.

see in Figure fig:scalevar. A scale-free graph is a network which has a degree distribution following a power law, $P(k) \sim k^{-\gamma}$, where k is the degree and γ is a constant chosen to best fit the observed data. These types of graphs are of interest because many real world networks such as social networks, the internet, and semantic networks [14, 15, 16], are thought to be networks of this type. Here the spectral method outperforms all other approaches, with the structural methods preforming the worst. The final two graphs in Figures 4.7–4.6 are small social network graphs, showing a wikipedia voting network, and a p2p gnutella file sharing network. In both social network experiments we see the structural methods perform poorly, while the hierarchical spectral method works well for both graphs.

All these graphs have the advantage that they are small enough to be directly inverted so that the actual error and variance may be determined. In all the example graphs the spectral methods tend to preform well, while probing and other structural methods tend to preform poorly on the less structured graphs, especially when it comes to variance

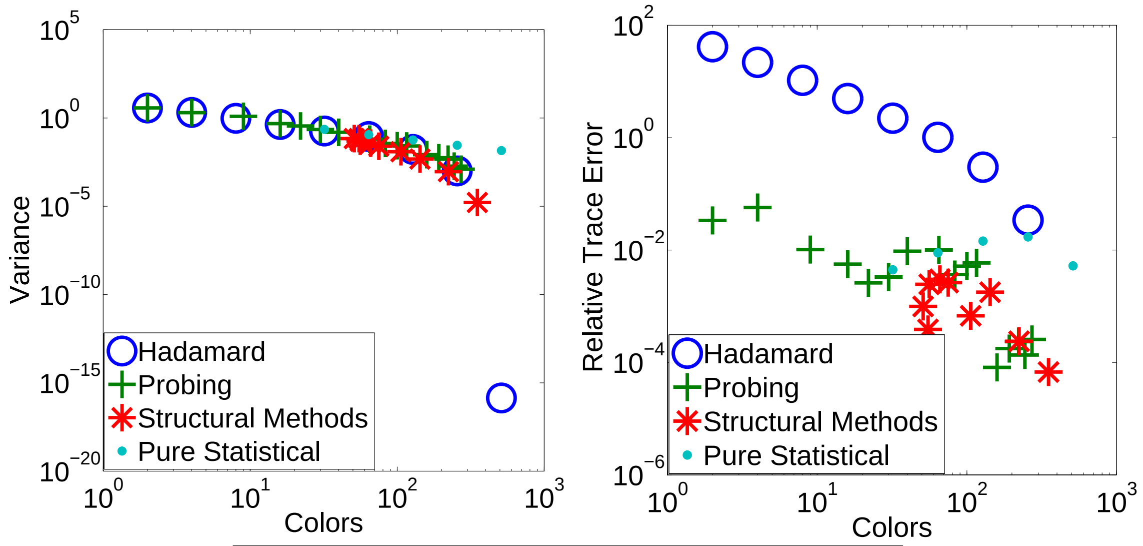


Density Range	Vs	Diagonals	Probing Powers
.1 − .92	50	1	4

Figure 4.7: The variance and relative trace error of the trace estimator for the p2p-gnutella05 graph.

reduction. More importantly, the variance of the spectral method tends to reduce faster than the Monte Carlo method, which shows promise for developing approximate spectral algorithms that have a similar effect.

Our structural methods have better success on the second set of matrices than they do on the graph Laplacian test cases. In general, they provide an improvement over probing and purely statistical methods. In the gre512 and orsreg test cases, the structural methods perform as well as or slightly better then probing, probably because both methods are finding the same dominant off-diagonals. The mhd14 matrix achieves similar results as probing for the range of colors it explores, but terminates early, because even with very low amounts of sparsification, the algorithm can detect very little structure. The nos6 and bcsstk07 matrices start off as being comparable to probing, but start to surpass it significantly as the density increases. The af23560 matrix is unique among the test cases because the structural methods never manage to surpass probing. However, the af23560 matrix clearly has little structure to make use of, since statistical methods, probing, Hadamard,



Density Range	Vs	Diagonals	Probing Powers
.1 − .9	50	418	16

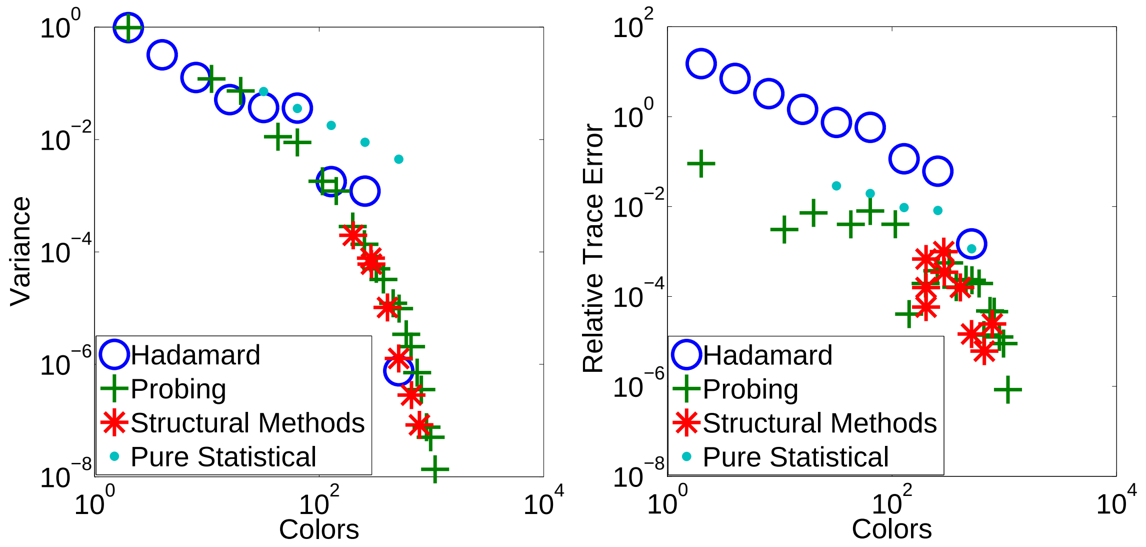
Figure 4.8: The variance and relative trace error of the trace estimator for the gre512 matrix.

and our method all preform the same.

Overall, our results indicate that our proposed structural methods can provide an improvement over probing at relatively low cost, even for matrices where probing should be expected to preform very well, that is matrices with a sharp decay away from the diagonal.

4.6 Conclusions

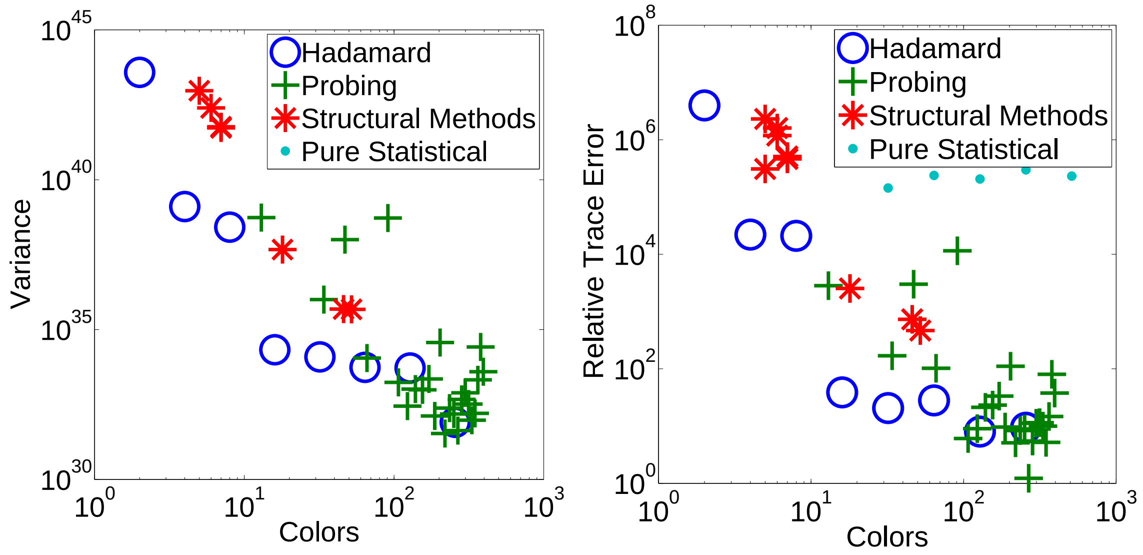
We have presented two classes of methods for estimating the trace of the inverse of an arbitrary matrix, which we have experimentally shown to be superior to both probing and purely statistical methods. The first class of these methods seeks to detect and exploit structural features of the matrix, and the second class seeks to use spectral information of the matrix to leverage ideas from algorithms designed for community detection and partitioning. We have also identified several promising areas of future research. In the



Density Range	Vs	Diagonals	Classical Probing Powers
$.1 - .99$	50	235	16

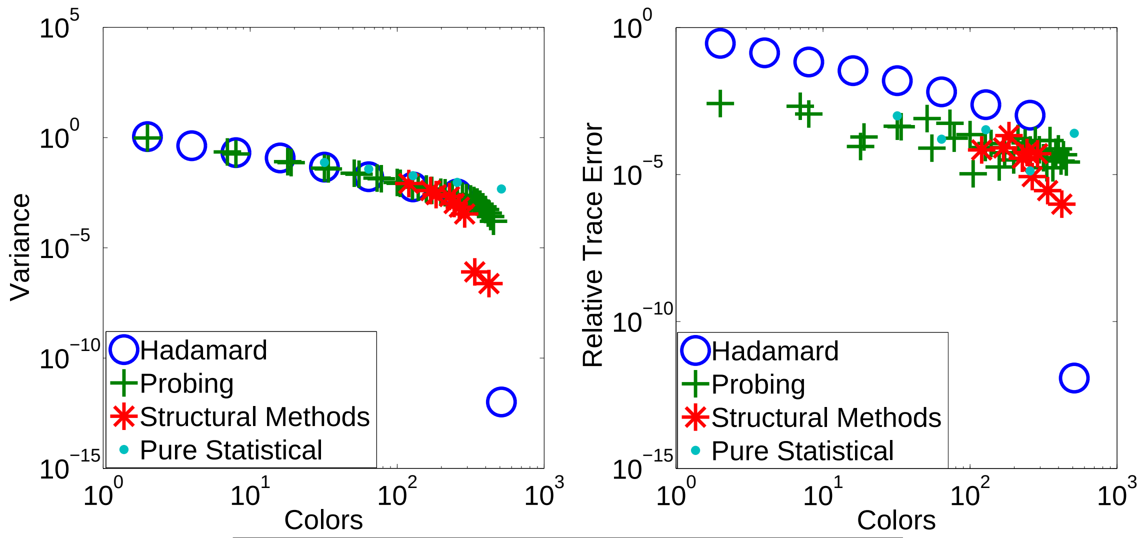
Figure 4.9: The variance and relative trace error of the trace estimator for the orsreg matrix.

structural arena, the prospect of leveraging research in adaptive sampling for use with the Nyström method may make these types of approaches more useful, due to the potential to use some of the theoretical error bounds these ideas provide, making the structural results more reliable. In the spectral arena, the challenge is to find algorithms that approximate the recursive spectral bisection of Algorithm 14 efficiently. Moreover, there are many other classes of spectral clustering algorithms which we have not yet experimented with that could yield improved algorithms for finding good colorings.



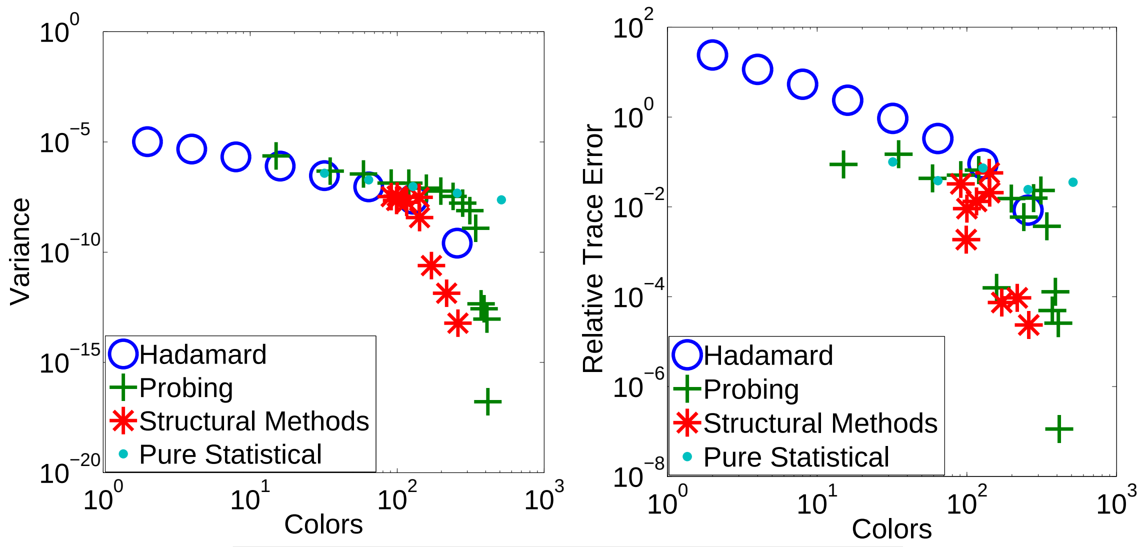
Density Range	Vs	Diagonals	Probing Powers
.1 − .999	50	115	16

Figure 4.10: The variance and relative trace error of the trace estimator for the mhd416 matrix.



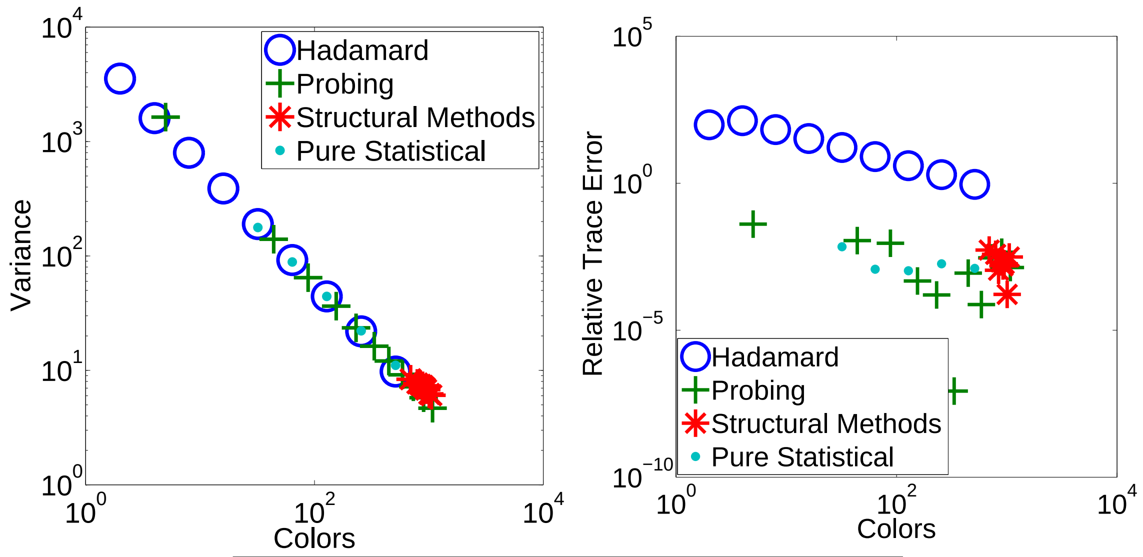
Density Range	Vs	Diagonals	Probing Powers
.1 − .999	50	181	16

Figure 4.11: The variance and relative trace error of the trace estimator for the nos6 matrix.



Density Range	Vs	Diagonals	Probing Powers
.1 − .999	50	248	16

Figure 4.12: The variance and relative trace error of the trace estimator for the bcsstk07 matrix.



Density Range	Vs	Diagonals	Probing Powers
.3 − .9	50	2623	16

Figure 4.13: The variance and relative trace error of the trace estimator for the af23560 matrix.

Chapter 5

Conclusion and future work

In this work we have investigated ways in which $\mathbf{Diag}(f(A))$ may be computed when information about the structure of A is known and we have explored algorithms for discovering such information when it is not known a priori. Specifically, we have made use of geometric, structural, and spectral methods to improve on the results of probing. We have also developed a method for combing this information with the statistical methods of [22] in order to obtain unbiased methods that provide also statistical error bounds on the estimates. Further, we have provided a framework for analyzing which matrices are likely to see improvement over purely statistical methods, based on how rapidly the largest elements of the matrix decays. Finally we have preformed experiments on many different tests cases, ranging from applications in LQCD, social network graphs, engineering, and PDEs. In some cases we obtained an order of magnitude speed up compared to previously known methods.

5.1 Methods for Lattices

Our methods greatly improved prior methods for estimating $\mathbf{Diag}(f(A))$ in matrices arising from PDEs discretized onto lattices. We have introduced two methods: an extremely efficient algorithm based on binary arithmetic which works when the dimensions of the matrix can be factored solely into powers of two, and another method which is more general. For real world cases arising in LQCD, we demonstrate significant improvements over prior methods for challenging cases, speeding up the statistical process by an order of magnitude.

5.2 Methods for General Matrices

Further improvement to classical probing appears difficult, due to the limitations of relying on a polynomial of A to approximate the structure of $f(A)$. To circumvent this issue we introduced several methods that seek to approximate the structure of $f(A)$ directly, bypassing the need to compute high order matrix polynomials. There are two main classes of information we use to discover the structure of $f(A)$. The first of these is structural information. By solving for random columns v of $f(A)$, we are able to form two approximations to $f(A)$. First, by looking at the cross-correlation of the vectors at specific lags, we can locate the prominent off-diagonals of $f(A)$. Second, we can form a rough approximation to $f(A)$ by finding vv^T. In many experiments we have found that these two methods taken together provide more accurate colorings than classical probing. Finally, we also make use of the spectrum of A to find good colorings of $f(A)$ in a process analogous to that used to find clusters in graphs.

Both these methods suggest future avenues of research. The structural methods have close parallels with the Nyström algorithm for sparse matrix approximation. Leveraging some of the approaches for sampling from this method may provide significant improvements to our algorithm. It may also be possible to apply the Nyström algorithm directly, an approach that deserves further study. The spectral methods also have many interesting open questions. First, significant research is still needed to make this algorithm practical. Additionally there are four other algorithmic ideas used for the graph clustering problem, which we have not yet tried to adapt to our problem. Finally, while we have achieved good results with this method for graph Laplacians, it remains to be seen if there is a way to extend this approach to more general classes of matrices.

Bibliography

[1] M. Benzi, E. Estrada, and C. Klymko,*Ranking Hubs and Authorities Using Matrix Functions, Linear Algebra and its Applications*, SIAM. J. Matrix Anal. Appl. 36 (2), pp. 686706 (2015)

[2] P. Drineas, R. Kannan, and M. W. Mahoney,*Fast monte carlo algorithms for matrices I: approximating matrix multiplication*, SIAM Journal on Computing, 36(1), pp. 132- 157 (2006)

[3] Shusen Wang and Zhihua Zhang, *Improving CUR Matrix Decomposition and the Nystrom Approximation via Adaptive Sampling*, Journal of Machine Learning Research (JMLR), 14: 2729-2769, (2013)

[4] J. Barlow J. Demmel, *Computing Accurate Eigensystems Of Scaled Diagonally Dominant Matrices SIAM J. Numer. Anal., v. 27, n. 3, pp. 762-791, (1990)*

[5] Y. Hou*,* Bounds for the least Laplacian eigenvalue of a signed graph Acta Mathematica Sinica, Volume 21, Issue 4, pp 955-960 (2005)

[6] Fang Chung, *Spectral Graph Theory*, CBMS Regional Conference Series in Mathematics, No. 92, (1996)

[7] A.Pothen, H. Simon, and K. Liou *Partitioning Sparse Matrices with Eigenvectors of Graphs* SIAM. J. Matrix Anal. Appl., 11(3), 430452. (1990)

[8] C. Alpert, A. Kahng, and S. Yao *Spectral partitioning with multiple eigenvectors* Discrete Appl. Math. 90, 1-3 (January 1999)

[9] F. Bauer, J. Jost *Bipartite and neighborhood graphs and the spectrum of the normalized graph Laplacian Communications in Analysis and Geometry Comm. Anal. Geom. 21 no. 4, 787-845, (2013)*

[10] S. Liu Multi-way dual Cheeger constants and spectral bounds of graphs Advances in Mathematics, Volume 268 , 306338 (2015)

[11] F. McSherry, *Spectral partitioning of random graphs FOCS 01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science, 529, (2001)*

[12] R. Boisvert, R. Pozo, K. Remington, R. Barrett, and J. Dongarra Matrix market: a web resource for test matrix collections, Proceedings of the IFIP TC2/WG2.5 working conference on Quality of numerical software: assessment and enhancement, 125-137. (1997)

[13] A. Taylor and D.J. Higham *CONTEST: A Controllable Test Matrix Toolbox for MATLAB ACM Transactions on Mathematical Software, 35 (4). 26:1-26:17, (2009)*

[14] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee Measurement and Analysis of Online Social Networks *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC '07). ACM, New York, NY, USA, 29-42,(2007)*

[15] L. Li, D. Alderson, J. Doyle, and W. Willinger Towards a Theory of Scale-Free Graphs: Definition,Properties, and Implications Internet Mathematics Volume 2, Number 4, 431-523, (2005)

[16] M. Steyvers J. Tenenbaum *The Large-Scale Structure of Semantic Networks: Statistical Analyses and a Model of Semantic Growth, Cognitive Science 29 (1): 4178, (2005)*

[17] T. F. Coleman and J. J. More, Estimation of sparse Jacobian matrices and graph coloring problems, SIAM Journal on Numerical Analysis, 20, pp. 187209,(1983)

[18] C. Siefert and E. de Sturler *Probing methods for generalized saddle-point problems , Electronic Transactions on Numerical Analysis, 22 , pp. 163183,(2006)*

[19] G. Golub and C. Van Loan Matrix computations 3rd ed. Johns Hopkins University Press (1996)

[20] K. Ahuja, B. Clark, E. de Sturler, D. M. Ceperley, and J. Kim, *Improved scaling for Quantum Monte Carlo on insulators*, (7 May 2011).

[21] P. R. Amestoy, I. S. Duff, Y. Robert, F.-H. Rouet, and B. Ucar, *On computing inverse entries of a sparse matrix in an out-of-core environment*, Tech. Rep. TR/PA/10/59, CERFACS, Toulouse, France, 2010.

[22] H. Avron and S. Toledo, *Randomized algorithms for estimating the trace of an implicit symmetric positive semi-denite matrix*, Journal of the ACM, 58 (2011), p. Article 8.

[23] R. Babich, R. Brower, M. Clark, G. Fleming, J. Osborn, C. Rebbi, and D. Schaich, *Exploring strange nucleon form factors on the lattice*, (4 May 2011).

[24] Z. Bai, M. Fahey, and G. H. Golub, *Some large-scale matrix computation problems*, Journal of Computational and Applied Mathematics, 74 (1996), pp. 71–89.

[25] G. S. Bali, S. Collins, and A. Schaefer, *Effective noise reduction techniques for disconnected loops in Lattice QCD*, (2010).

[26] M. Beck and S. Robins, *Computing the Continuous Discretely: Integer-Point Enumeration in Polyhedra*, Springer, 2007.

[27] C. Bekas, A. Curioni, and I. Fedulova, *Low cost high performance uncertainty quantication*, in In WHPCF 09: Proc. of the 2nd Workshop on High Performance Computational Finance, New York, NY, USA, 2009, ACM, pp. 1–8.

[28] C. Bekas, E. Kokiopoulou, and Y. Saad, *An estimator for the diagonal of a matrix*, Appl. Numer. Math., 57 (2007), pp. 1214–1229.

[29] M. Benzi, P. Boito, and N. Razouk, *Decay properties of spectral projectors with applications to electronic structure*, SIAM Review, (to appear).

[30] M. Benzi and G. H. Golub, *Bounds for the entries of matrix functions with applications to preconditioning*, BIT, 39 (1999), pp. 417–438.

[31] S. Bernardson, P. McCarty, and C. Thron, *Monte Carlo methods for estimating linear combinations of inverse matrix entries in lattice QCD*, Comput. Phys. Commun., 78 (1994), pp. 256–264.

[32] M. Blaum and J. Bruck, *Interleaving schemes for multidimensional cluster errors*, IEEE Transactions on Information Theory, 44 (1998), pp. 730–743.

[33] D. Bozdag, U. Catalyurek, A. Gebremedhin, F. Manne, E. Boman, and F. Ozguner, *Distributed-memory parallel algorithms for distance-2 coloring and related problems in derivative computation*, SIAM J. Sci. Comput, 32 (2010), pp. 2418–2446.

[34] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iteration*, SIAM J. Sci. Statist. Comput., 19 (1998), pp. 995–1023.

[35] T. F. Coleman and J. J. Moré, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM Journal on Numerical Analysis, 20 (1983), pp. 187–209.

[36] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, USA, 1989.

[37] J. Foley., K. J. Juge, A. O'Cais, M. Peardon, S. Ryan, and J.-I. Skullerud, *Practical all-to-all propagators for lattice qcd*, Comput. Phys. Commun., 172 (2005), pp. 145–162.

[38] A. H. Gebremedhin, F. Manne, and A. Pothen, *What color is your Jacobian? Graph coloring for computing derivatives*, SIAM Rev., 47 (2005), pp. 629–705.

[39] G. H. Golub and G. Meurant, *Matrices, moments and quadrature*, in Numerical Analysis 1993, D. Griffiths and G. Watson, eds., vol. 303, Longman Scientific & Technical, Pitman Research Notes in Mathematics Series, 1994.

[40] H. Guo, *Computing traces of functions of matrices*, Numerical Mathematics, A Journal of Chinese Universities (English series), 2 (2000), pp. 204–215.

[41] H. Guo and R. Renaut, *Estimation of $u^t f(a)v$ for large-scale unsymmetric matrices*, Numerical Linear Algebra with applications, 11 (2004), pp. 75–89.

[42] R. Gupta, *Introduction to Lattice QCD.* arXiv:hep-lat/9807028v1 [http://arxiv.org/abs/hep-lat/9807028], 1998.

[43] K. J. Horadam, *Hadamard matrices and their applications*, Princeton University Press, 2006.

[44] T. Huckle, *Approximate sparsity patterns for the inverse of a matrix and preconditioning*, Appl. Numer. Math., 30 (1999), pp. 291–303.

[45] M. F. Hutchinson, *A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines*, J. Commun. Statist. Simula., 19 (1990), pp. 433–450.

[46] T. Iitaka and T. Ebisuzaki, *Random phase vector for calculating the trace of a large matrix*, Phys. Rev. E, 69 (2004), p. 05770110577014.

[47] I. C. Ipsen and D. J. Lee, *Determinant approximations*, Tech. Rep. TR 03-30, North Carolina State University, Department of Mathematics, 2003.

[48] D. J. Lee and I. C. F. Ipsen, *Zone determinant expansions for nuclear lattice simulations*, Phys. Rev. C, 68 (2003), p. 064003.

[49] L. Lin, J. Lu, L. Ying, R. Car, and W. E, *Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems*, Commun. Math. Sci., 7 (2009), pp. 755–777.

[50] L. Lin, C. Yang, J. C. Meza, J. Lu, L. Ying, and W. E., *Selinv–an algorithm for selected inversion of a sparse symmetric matrix*, ACM Transactions on Mathematical Software, 37 (4), pp. Article 40, pages 19.

[51] C. Morningstar, J. Bulava, J. Foley, K. Juge, D. Lenkner, M. Peardon, and C. Wong1, *Improved stochastic estimation of quark propagation with Laplacian Heaviside smearing in lattice QCD*, Phys. Rev. D, 83 (2011).

[52] F. Pukelsheim, *Optimal design of experiments*, SIAM, Classics in Applied Mathematics. 50., 1993.

[53] A. Reusken, *Approximation of the determinant of large sparse symmetric positive definite matrices*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 799–818.

[54] H. J. Rothe, *Lattice Gauge Theories: An introduction*, World Scientific Publishing Co. Pte. Ltd., 2005.

[55] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, 2nd edition, Philadelphia, PA, USA, 2003.

[56] B. Sheikholeslami and R. Wohlert, *Improved Continuum Limit Lattice Action for QCD with Wilson Fermions*, Nucl.Phys., B259 (1985), p. 572.

[57] C. Siefert and E. de Sturler, *Probing methods for generalized saddle-point problems*, Electronic Transactions on Numerical Analysis, 22 (2006), pp. 163–183.

[58] Z. Strakos and G. H. Golub, *Estimates in quadratic formulas*, Numerical Algorithms, 8 (1994), pp. 241–268.

[59] Z. Strakos and P. Tichy, *On efficient numerical approximation of the bilinear form $c^*a^{-1}b$*, SIAM J. Sci. Comput., 33 (2011), pp. 565–587.

[60] J. Tang and Y. Saad, *Domain-decomposition-type methods for computing the diagonal of a matrix inverse*, Report UMSI 2010/114.

[61] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics (2003)

[62] ——, *A probing method for computing the diagonal of the matrix inverse*, Report UMSI 2010/42.

[63] L. R. Welch, *Lower bounds on the maximum cross correlation of signals*, IEEE Trans. on Info. Theory, 20 (May 1974), pp. 397–399.

[64] K. G. Wilson, *Confinement of quarks*, Phys. Rev., D10 (1974), pp. 2445–2459.

[65] M. N. Wong, F. J. Hickernell, and K. I. Liu, *Computing the trace of a function of a sparse matrix via Hadamard-like sampling*, Tech. Rep. 377(7/04), Hong Kong Baptist University, 2004.

[66] A. Stathopoulos, J. Laeuchli, and K. Orginos, *Hierarchical probing for estimating the trace of the matrix inverse on toroidal lattices* , SIAM J. Sci. Comput.,35(5) (2013), pp. 299–322.

[67] F. Rouet, *Calcul partiel de l'inverse d'une matrice creuse de grande taille - application en astrophysique*, Master's Thesis (10/09), Institut National Polytechnique de Toulouse, 2009.

[68] E. Estrada, N. Hatano, *Statistical-mechanical approach to subgraph centrality in complex networks*, Chemical Physics Letters, 439, (5/07), pp. 247-251.

[69] N. Higham, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, 2008.

[70] S. Li, E. Darve, *Extension and optimization of the FIND algorithm: Computing Green's and less-than Green's functions*,J. Comput. Physics 231(4), (5/12), pp. 1121-1139.

[71] S. Li, E. Darve, *Some new mathematical methods for variational objective weather analysis using splines and cross-validation*,Mon. Weath. Rev. 108, (6/80), pp. 1122-1143

[72] S. Li, E. Darve, *Smoothing noisy data with spline functions*,Numer. Math. 31, (2/79), pp. 377-403

[73] S. W. Golomb and L. D. Baumert, *The search for Hadamard matrices*, Amer. Math. Monthly, 70 pp. 12-17 (1963)

[74] P. Drineas , R. Kannan , M. Mahoney, *Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication*,SIAM Journal on Computing 36, (9/04), pp. 132157

[75] D. Chen, S. Toledo, *Vaidya's preconditioners: Implementation and experimental study*,Electronic Transactions on Numerical Analysis, 16,(9/03), pp. 30-49