Optimizing Tensor Contractions for Nuclear Correlation Functions

by

Pranjal Vachaspati

Submitted to the Department of Physics
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Physics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

Signature redacted

Author ....
Department of Physics
May 9, 2014

Signature redacted

Certified by ....
William Detmold
Assistant Professor of Physics
Thesis Supervisor

Signature redacted

Accepted by ....
Nergis Mavalvala
Senior Thesis Coordinator, Department of Physics

Optimizing Tensor Contractions for Nuclear Correlation Functions

by

Pranjal Vachaspati

Submitted to the Department of Physics
on May 9, 2014, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Physics

Abstract

Nuclear correlation functions reveal interesting physical properties of atomic nuclei, including ground state energies and scattering potentials. However, calculating their values is computationally intensive due to the fact that the number of terms from quantum chromodynamics in a nuclear wave function scales exponentially with atomic number.

In this thesis, we demonstrate two methods for speeding up this computation. First, we represent a correlation function as a sum of the determinants of many small matrices, and exploit similarities between the matrices to speed up the calculations of the determinants. We also investigate representing a correlation function as a sum of functions of bipartite graphs, and use isomorph-free exhaustive generation techniques to find a minimal set of graphs that represents the computation.

Thesis Supervisor: William Detmold
Title: Assistant Professor of Physics

Acknowledgments

Most of all, I would like to thank Professor Will Detmold. He has been an incredible mentor, teacher, and adviser. I've learned a huge amount from him over the past two years, both about physics and about how to do research. He has helped me make the most of my time in the physics department, and has been unbelievably patient with me throughout that time.

I would like to thank my father, who taught me math that I didn't yet know I wanted to know, and my mother, who gave me lots of books to read as a child.

I want to thank my friends on Putz and elsewhere for supporting me these last four years. I couldn't have made it through here without you.

Contents

List of Figures

Chapter 1

Lattice QCD

1.1 The Discretized Path Integral

The path integral formulation of quantum mechanics indicates that the propagator of some quantum system can be written as:

$$\langle \psi(x_0, t_0) | \psi(x_1, t_1) \rangle = \langle x_0 | e^{-H(t_1 - t_0)} | x_1 \rangle = \int \mathcal{D}x(t) e^{-S[x]} \tag{1.1}$$

Here, $\mathcal{D}x(t)$ is an integral over all possible paths the particle can take between x_0 and x_1. $S[x]$ is the action of a path. The classical action for a particle moving in a potential V is

$$S[x] = \int_{t_0}^{t_1} dt L(x, \dot{x}) = \int_{t_0}^{t_1} dt \left[\frac{1}{2} m x'(t)^2 + V(x(t)) \right] \tag{1.2}$$

This formulation is continuous, so to put it on a computer it must be discretized. A four dimensional lattice is used, with three spatial coordinates and one time coordinate, n elements along each lattice dimension, and spacing a. A given path $x(t)$ becomes a n-element vector with one element in each time slice. Now, the action is a sum, with a discretized derivative:

$$S[x] = \sum_{i=0}^{n} \frac{1}{2} x_i'^2 + V(x_i) \tag{1.3}$$

The derivative can be discretized based on the Taylor expansion of $x(t)$:

$$x(t+a) = x(t) + \frac{1}{1!}x'(t)a + \frac{1}{2!}x''(t)a^2 + \frac{1}{3!}x'''(t)a^3 + \dots \tag{1.4}$$

The simplest discretization of the derivative considers the first two terms in the expansion:

$$x(t+a) = x(t) + \frac{1}{1!}x'(t)a + O(a^2) \tag{1.5}$$

$$x'(t) = \frac{x(t+a) - x(t)}{a} + O(a) \tag{1.6}$$

This has an error term linear in a. By including higher-order terms from the Taylor expansion, this term can be reduced to an arbitrary power of a.

The integral over all paths becomes an integral over the value of x at each lattice time slice:

$$\int \mathcal{D}x(t) \rightarrow \int dx_1 dx_2 \dots dx_{n-1} \tag{1.7}$$

Since the start and end positions are held fixed, there is no need to integrate over them.

After discretizing in space, the path integral becomes a sum over each of the time slices:

$$\langle x_0 | e^{-H(t_1-t_0)} | x_1 \rangle = \sum_{x_1} \sum_{x_2} \dots \sum_{x_{n-1}} e^{-S[\{x_0, x_1, x_2, \dots, x_n\}]} \tag{1.8}$$

1.2 The Monte Carlo Integral

Unfortunately, computing Equation 1.8 via quadratures is incredibly inefficient, requiring n^{3n} evaluations of the action. Instead, a randomized Monte Carlo process can be used to estimate the propagator value.

In particular, the expectation value of an observable defined as a functional over a given path $\Gamma[x]$ is given by

$$\langle \Gamma \rangle = \sum_{[x]} \Gamma[x] e^{-S[x]} \tag{1.9}$$

Instead of generating all paths and measuring the action, the Metropolis algorithm, described in [10], generates paths with probability

$$P([x]) = e^{-S[x]}. \tag{1.10}$$

Therefore, given a set of paths $\{[x]\}$ generated in this way, the expectation value of Γ can be approximated by simply summing over $\Gamma[x]$.

To evaluate nuclear observables, we need to consider quantum chromodynamics on the lattice. The conceptual picture described above stays the same, but instead of particle paths, observables are computed over configurations of quark and gluon fields. Furthermore, the classical action is no longer used; instead, the classical QCD action is approximated using combinations of loops on the lattice, also described in [10]. In addition, there are many more degrees of freedom, since each lattice point is no longer a scalar, but a tensor with spin, color, and flavor indices.

1.3 Nuclear Creation and Annihilation Operators

Here, we focus on the problem of evaluating specific operators on a single field configuration in a given volume. The eventual goal is to find ground state energies of atomic nuclei, which can be extracted from the large t limit of $\langle \bar{\mathcal{N}}_h(0)\mathcal{N}_h(t)\rangle$:

$$\langle \bar{\mathcal{N}}_h(0)\mathcal{N}_h(t)\rangle = \langle \bar{\mathcal{N}}_h|e^{-Ht}|\mathcal{N}_h\rangle = \sum_n \langle \bar{\mathcal{N}}_h|n\rangle e^{-E_n t}\langle n|\mathcal{N}_h\rangle \tag{1.11}$$

where $\mathcal{N}_h$ is a nuclear creation operator built from quark and gluon fields described by quantum numbers h. In the large t limit, all non-ground state terms go to zero because of the exponential damping and we can recover the ground state energy of the system.

First, we define the nuclear creation operator $\bar{\mathcal{N}}_h$ in terms of quark creation operators. If $\mathcal{N}_h$ has A nucleons and n_q quarks, it can described as a tensor product of individual quark creation operators, taking the form

$$\bar{\mathcal{N}}_h = \sum_{\{a\}} w_h^{a_1 \dots a_{n_q}} \bar{q}(a_1)\bar{q}(a_2)\dots\bar{q}(a_{n_q}) \tag{1.12}$$

where $\bar{q}(a_1)$ is a quark creation operator for a quark with quantum numbers represented by a_1. w_h is an antisymmetric tensor that is nonzero only when none of the indices are equal and the combination of the quark quantum numbers is equal to the nuclear quantum number [4].

If there are N possible quark indices, the number of non-zero terms in the sum is

$$\frac{N!}{n_q!(N - n_q!)} \tag{1.13}$$

after accounting for permutations. A number of strategies, described in [1], can reduce this number by taking into account various physical symmetries and by considering only simple spatial wave functions.

For example, for an alpha particle, which has four nucleons and twelve quarks, we consider a wave function where all particles are on the same site, a multi-site wave function with two deuterons, a multi-site wave function with a diproton and a dineutron, and a multi-site wave function with a triton and a proton. Each of these wave functions has a different number of terms. Since there are twelve quarks in an alpha particle, and there can be only twelve quarks on a single site, the wave function with all particles on the same site has only one term. The diproton-dineutron and triton-proton wave functions must be rotationally symmetric, so they have 5718 and 1944 terms respectively.

1.4 Nuclear Correlation Functions

To actually find the nuclear ground state energy, we must find the correlation function between two nuclear wave functions, a source and a sink, of the form

$$\langle \mathcal{N}_1(t) | \bar{\mathcal{N}}_2(0) \rangle \tag{1.14}$$

which is an integral over the gauge fields and quark configurations of each wave function. For a specific configuration of gluon degrees of freedom, this is given by

$$\mathcal{N}_1(t)\bar{\mathcal{N}_2}(0) = \int \mathcal{D}q\mathcal{D}\bar{q}e^{-S}(\sum_{\{a\}} w_1^{a_1 \dots a_{n_q}} \bar{q}(a_1)\bar{q}(a_2)\dots\bar{q}(a_{n_q}))(\sum_{\{a\}} w_2^{a_1 \dots a_{n_q}} \bar{q}(a_1)\bar{q}(a_2)\dots\bar{q}(a_{n_q})) \tag{1.15}$$

This can be rephrased in terms of quark propagator $S(q(a_0), q(a_1); t)$ as

$$\mathcal{N}_1(t)\bar{\mathcal{N}_2}(0) = e^{-S}\sum w_1 w_2 \sum_i \sum_j S(q(a_{i1}), q(a_{j1}))S(q(a_{i2}), q(a_{j2}))\dots S(q(a_{in_q}), q(a_{jn_q}))\epsilon^i\epsilon^j \tag{1.16}$$

where ϵ^i and ϵ^j ensure that the indices are fully antisymmetrized.

Fast evaluation of this sum is critical. Even moderately complex wave functions have thousands of terms, so the total number of terms that needs to be calculated is in the tens or hundreds of millions. Each of these calculations must be done on a number of gauge configurations, so even correlators made from simple wave functions quickly become difficult to compute.

1.5 Correlator Graphs

The most natural way to represent these sums is as a set of bipartite graphs, as in Figure 1-1. Hadrons in a given pair of sink-source terms are represented as nodes on the right or left sides of a graph. Edges are allowed between two nodes only if one is in the sink and the other is in the source, and if the hadrons share a quark with the same flavor. Further, hadrons are constrained to have the correct number of edges for each flavor they contain (a proton has two up edges and one down edge, for example). Chapter 3 describes implementation of an isomorph-free generation algorithm for these graphs and its implications.

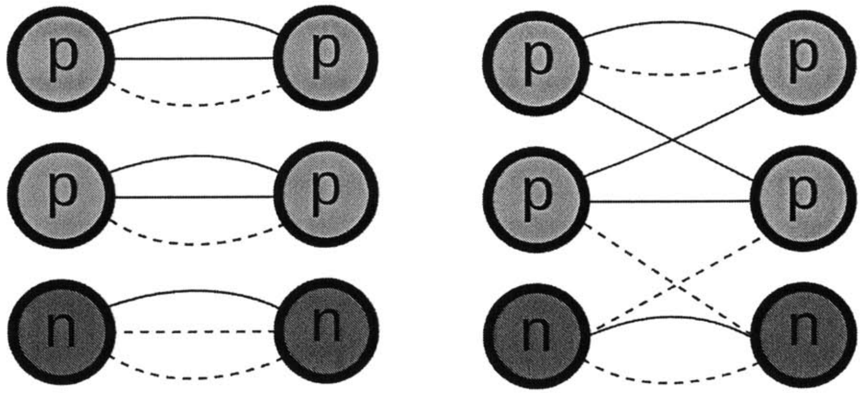


Figure 1-1: Graphs of two terms in a single-site triton-triton correlator. Dashed lines represent down quarks, solid lines represent up quarks.

1.6 Correlator Matrices And The Determinant Method

The sum over indices i and j in Equation 1.16 is equivalent to calculating the determinant of a matrix where the $(i,j)^{th}$ element is the correlator between the i^{th} source quark and the j^{th} sink quark [4]. The correlation function is then

$$\mathcal{N}_1(t)\bar{\mathcal{N}_2}(0) = e^{-S}\sum w_1 w_2 \det(G), \tag{1.17}$$

where

$$G_{ij} = S(q(a_i), q(a_j)). \tag{1.18}$$

Normally, this determinant can be calculated in $O(N_q^3)$ time by using the LU factorization of the matrix. Chapter 2 describes faster methods of finding the determinant using relationships between different pairs of sink-source terms.

Chapter 2

Fast Updates to LU Factorization

2.1 LU Factorization

The *LU* factorization of a matrix *M* finds an upper triangular matrix *U* and a unit lower triangular matrix *L* such that $M = LU$. The determinant of *M* is then the products of the determinants of *L* and *U*, which can be quickly computed as the product of elements on the diagonals. For example, a 3×3 matrix factors into l and u as:

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} \tag{2.1}$$

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} = \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ l_{21}u_{11} & l_{21}u_{12} + u_{22} & l_{21}u_{13} + u_{23} \\ l_{31}u_{11} & l_{31}u_{12} + l_{32}u_{22} & l_{31}u_{13} + l_{32}u_{23} + u_{33} \end{pmatrix}. \tag{2.2}$$

This reveals a method for determining the elements of l and u:

$$u_{ij} = m_{ij} - \sum_{k=1}^{i} l_{ik}u_{kj} \tag{2.3}$$

$$l_{ij} = \frac{1}{u_{jj}} \left(m_{ij} - \sum_{k=1}^{j} l_{ik}u_{kj} \right) \tag{2.4}$$

There exist non-singular matrices that do not have an LU factorization of this form. For example, if

$$M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.5}$$

then $u_{11} = 0$, and there is no l_{21} such that $u_{11}l_{21} = 1$. Furthermore, in an numerical computation, even without zero entries, small values on the diagonal of U can cause the algorithm to become unstable, even on well-conditioned matrices.

However, any matrix that has all non-singular upper-left square matrices has a unique LU decomposition, determined by the procedure above. Furthermore, most[1] matrices can be transformed into a form suitable for LU decomposition by permuting the rows to maximize the magnitudes of the elements on the diagonals, a process known as pivoting. This gives a factorization $PA = LU$ where P is a permutation matrix.

2.1.1 Computational Cost

LU factorization requires $O(n^3)$ multiply-add operations. Specifically, computing the first row of U requires zero operations for n elements, computing the second row requires one operation each for $n-1$ elements, and so on; likewise, computing the first column of L requires one divide and zero multiply-adds for $n-1$ elements, the second column requires one divide and one multiply-add for $n-2$ elements, and so on. Therefore, the total number of operations is

$$\sum_{k=1}^{n} k(n-k) + \sum_{k=1}^{n}(k-1)(n-k) = \sum_{k=0}^{n}(kn-k^2) = \frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6} \tag{2.6}$$

fused multiply-adds and

$$\sum_{k=1}^{n}(n-k) = \frac{n^2}{2} - \frac{n}{2} \tag{2.7}$$

[1]In practice, this works for almost all matrices [16], but finding realistic counterexamples is a current area of research [7]

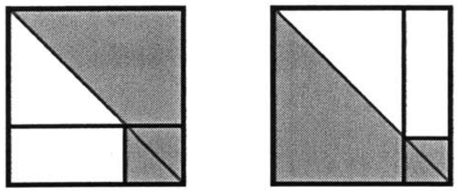

(a) Updated row (b) Updated column

Figure 2-1: Updating the i^{th} row or column of an LU factorization takes two phases. First, the i^{th} row of the factorization is calculated, using the part of the matrix shown in green. Then, the submatrix in blue is updated.

divides (alternatively, the reciprocals of the diagonal elements of U can be cached and these operations can be multiplies instead). Further, n multiplies are required to find the determinant given the LU factorization.

2.2 Determinant Updates

A number of matrices in the correlators described in Chapter 1 are very similar, differing only by a single row or column. This allows several $O(n^2)$ methods, and even some $O(n)$ methods, for finding the determinant of the matrix $M + uv^T$, where the LU factorization of M is known.

2.2.1 Optimality

No general method faster than $O(n^2)$ is possible. If it was, it would be possible to compute the determinant of an arbitrary matrix in time faster than $O(n^3)$ by performing n row updates[2]. However, as will be shown in Section 2.2.4, restricting the class of updates allows linear-time algorithms.

2.2.2 *LU* Update

The simplest update to an *LU* factorization simply recomputes all the terms that involve the modified elements, as seen in Figure 2-1. In the average case, this is faster than computing the factors from scratch, but it still has the same worst case complexity, which arises if the first row is modified. The cost of updating the actual changed column is of order $O(n^2)$, while the cost of updating the trailing submatrix (blue in Figure 2-1) is $O(k^3)$ where k is the dimension of that submatrix.

2.2.3 Bennett's Algorithm

Bennett [2] presents an algorithm which computes an arbitrary rank-1 update to an LU factorization, of the form $A' = A + uv^T$, in $O(n^2)$ time. The algorithm is modified [15] to proceed row-by-row for optimal memory access, and requires $4n^2$ operations to complete.

Both these approaches work on matrices that were originally factored with pivoting, but fails if the new row provides a poor pivot. When tested in practice, pivoting was found to be unnecessary, and numerical instability was not an issue, since the magnitude of the largest difference between the matrices factored with LAPACK and the matrices factored with the update algorithms was found to be less than one part in 10^8.

2.2.4 Matrix Determinant Lemma

Updating *LU* factorizations without repivoting is numerically unstable, and updating factorizations with repivoting is complicated. Instead, we can compute the determinant of the modified matrix without calculating the new LU factorization. This relies on the identity [6]

$$\det(A + uv^T) = (1 + v^T A^{-1} u) \det(A). \tag{2.8}$$

[2]It is actually possible to compute the determinant in $O(n^{2.3729})$ time, using the Coppersmith-Winograd algorithm for matrix multiplication, but this is of more theoretical than practical interest due to the large constant factors involved

For a row update, u is e_1; for a column update, v is e_1.

To prove this, we first see that

$$\det(I + uv^T) = (1 + v^T u) \tag{2.9}$$

For our purposes, it is sufficient to consider uv^T to be a single row or column. The only element of this that matters when calculating the determinant of $I + uv^T$ is the one on the diagonal, since the determinant of the matrix will be the product of the elements on the diagonal. From here, it is clear that

$$\det(A + uv^T) = \det(A)\det(I + A^{-1}uv^T) = \det(A)\det(I + v^T A^{-1}u) \tag{2.10}$$

$A^{-1}u$ is easily computed using the LU factorization, since the systems $Ux = b$ and $Ly = x$ can be solved in a total of $n^2 - n$ multiply-adds and $2n$ divides.

Somewhat surprisingly, it is possible to calculate some rank-1 updates in $O(n)$ time. If a parent matrix A has the i^{th} row changed in two different ways to get two new matrices, the first new determinant will be

$$\det(A + e_i v_1^T) = (1 + v_1^T A^{-1} e_i)\det(A), \tag{2.11}$$

and the second will be

$$\det(A + e_i v_2^T) = (1 + v_2^T A^{-1} e_i)\det(A). \tag{2.12}$$

The term $A^{-1}e_i$ corresponds to the i^{th} column of the inverse of A. This can be cached after the first computation so that the second computation requires only the computation of a vector-vector inner product.

Note that this does not violate the principle given in Subsection 2.2.1, since multiple rows cannot be changed without recalculating the inverse.

2.3 Finding Execution Strategies

Given an algorithm for updating *LU* factorizations, it is useful to find an evaluation order for the matrices that maximizes the number of matrices that can be computed with updates instead of from scratch. Given a set of matrices of lattice element lookups that correspond to a computation, we use the MapReduce-MPI library (`http://mapreduce.sandia.gov`) [14] to generate a graph where adjacent nodes represent matrices separated by a rank-1 update, using the map and reduce functions in Figure 2-2.

The optimal strategy for *LU* updates is very simple, since only one matrix in every connected component needs to be computed from scratch. Breadth-first search works well, because it allows matrices to be discarded quickly after they are first calculated.

On the other hand, if only determinant updates are used, the optimal strategy is to compute from scratch the matrices in the minimum dominating set of the graph of nodes. This is defined as the smallest set of vertices such that every vertex is adjacent to at least one member of the set. Unfortunately, computing this set is exponentially difficult, but a simple greedy algorithm provides a good approximation [13]. At each step, the vertex with the most edges is chosen, then it and its neighbors are removed from the graph.

2.4 Implementation and Results

A set of matrices corresponding to five terms in a source triton-proton wavefunction and all 1944 terms in a triton-proton sink wavefunction were computed on 1024 sites (a $4^3 \times 8$ lattice). When breadth-first search was used to find an execution strategy, 5120 matrices were computed from scratch, and the remaining 9939415 were calculated using the LU update method from Section 2.2.2. When the dominating set approach was used, 1605208 (17%) matrices were computed from scratch and 7781764 (83%) matrices were calculated with updating. When the inverse was cached in the determinant update, 2225577 (23%) determinants were calculated in $O(n^2)$ time

```
Map(m : Matrix)
for $i := 0$ ; $i < m.size$; $i = i + 1$ do
    $m' := m$;
    $m'[i, 0 : m.size] = 0$;
    Output(m', m);
end
for $i := 0$ ; $i < m.size$; $i = i + 1$ do
    $m' := m$;
    $m'[0 : m.size, i] = 0$;
    Output(m', m);
end
Reduce(M : [Matrix])
for $(m, n) \leftarrow M \times M$ do
    Output(m, n);
end
```

Figure 2-2: The map and reduce functions used to generate the graph of matrices

without caching, and 5556187 (60%) determinants were calculated in $O(n)$ time with caching. The total number of determinants calculated for the two approaches differs slightly due to differences in methods used to eliminate identical matrices. The times taken to calculate the determinants are given in Figure 2-3. The dominating set strategy requires substantially more matrices to be factored from scratch, but allows the very fast cached matrix determinant lemma algorithm to be used. This method is about six times as fast than the slowest method, which uses LAPACK to calculate the determinants from scratch.

2.4.1 Performance Enhancements

The program written to calculate the determinants is relatively unoptimized. A wide array of improvements could substantially enhance its performance. First of all, the code should be extremely parallelizable, since each parent node with its child nodes can be computed separately. Next, the vector units could be used more effectively. Currently, each vector instruction processes a single complex number, using 128 bits; however, AVX instructions on recent Intel processors support 256-bit wide instructions and the vector unit on a Xeon Phi processor is 512 bits wide, so this could

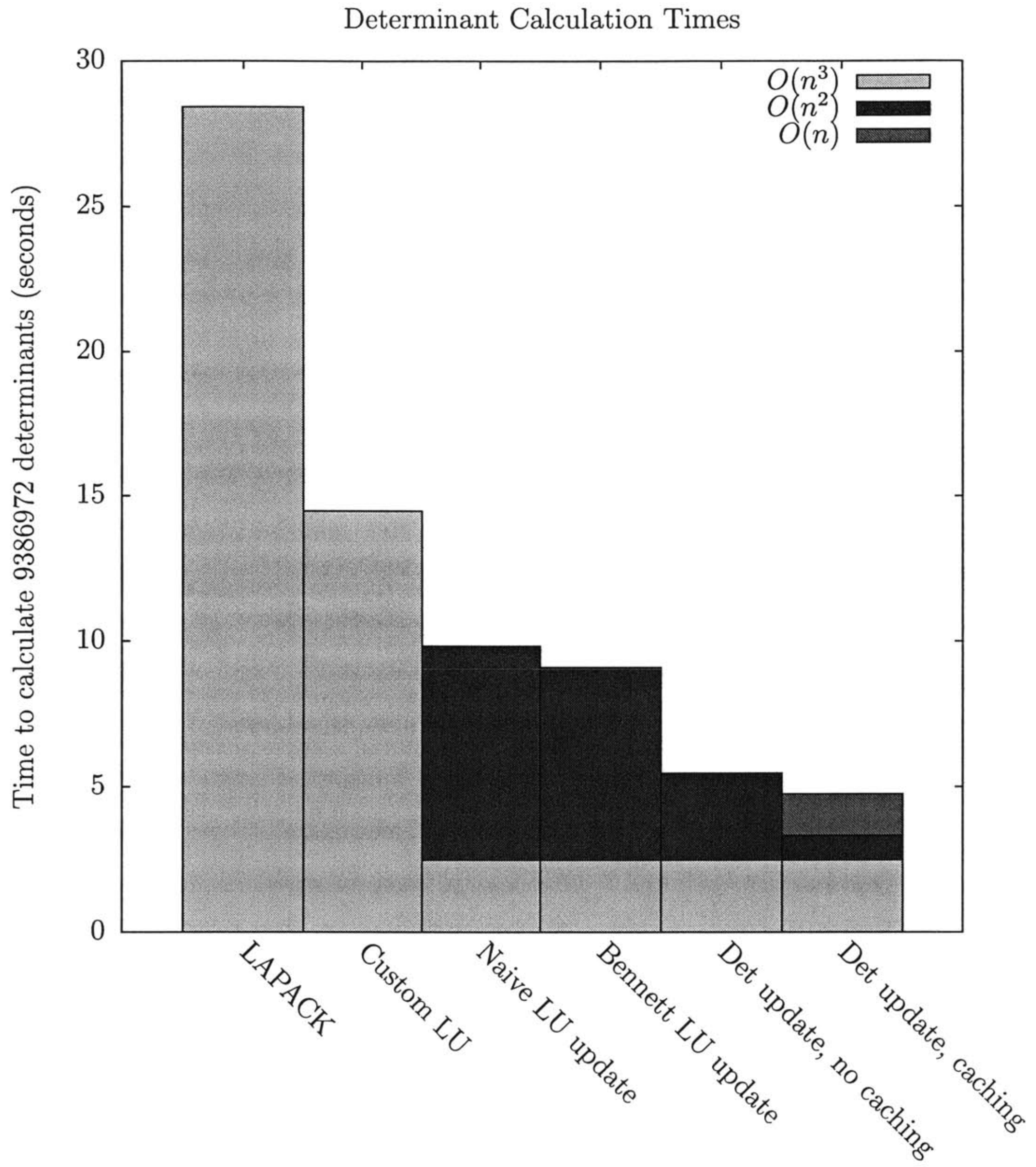


Figure 2-3: Time taken by the algorithms to find 9386972 determinants, and the time that each algorithm spent doing $O(n^3)$, $O(n^2)$, and $O(n)$ determinant calculations. Performance was measured on a single core of an 2.2 GHz Intel Xeon E5-4607 processor.

potentially give a two to four fold speedup. Additionally, improving the quality of the dominating set approximation could be extremely helpful. Currently, only 17% of the matrices are factored from scratch, but those calculations take up more than half of the running time. It should be possible to generate the matrices in an optimal order by considering the structure of terms in each wave function; if two terms differ by a single quark index, all the matrices involving those terms will differ by a single row or column. Finally, prefetching matrix terms used in a group of determinants could significantly reduce the time spent in memory accesses.

Chapter 3

Graph Representation of Contractions

The graphs described in Section 1.5 are represented as a set of vertices that correspond to hadrons, and edges between them that correspond to the selection of a particular set of quark propagators.

This structure is known as a colored multigraph, which is a graph that allows multiple colored edges between colored vertices. Edges that correspond to the same flavor (up, down, strange) have the same color, and vertices that have the same color have the same quark constituents and are both in the sink wave function or both in the source wave function.

Our graphs have a fixed number of vertices; we will build them by adding edges. Call G with the added edge between v_i and v_j *augmented*, and refer to it as $\{G, (v_i, v_j)\}$. The source half of G is denoted by G_{src} while the sink half is denoted by G_{snk}.

3.1 Graph Isomorphisms and Automorphism Groups

Finding isomorphic graphs for a term in a correlator reveals important information about the structure of that correlator, and indicates similarities between graphs that can potentially be leveraged to reduce the amount of computation needed to evaluate the correlator.

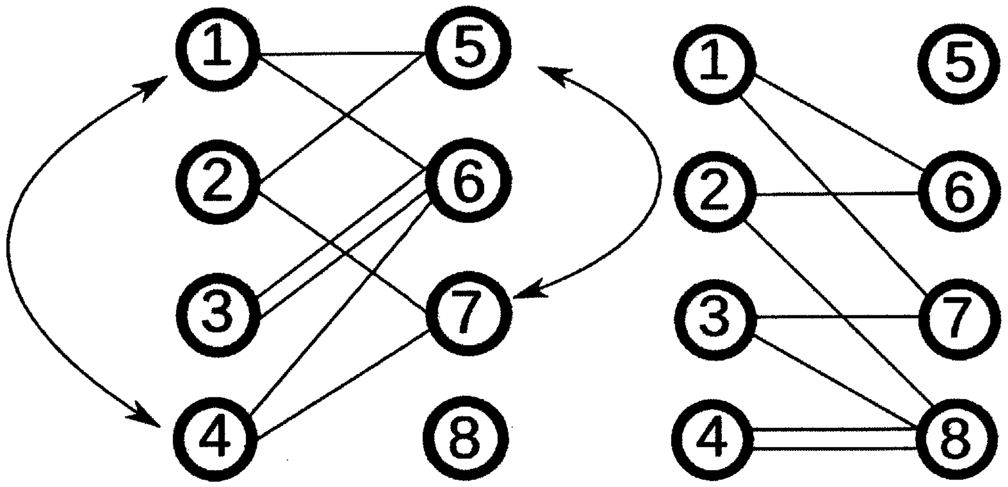


Figure 3-1: Two isomorphic graphs. An automorphism of the graph on the left is given by the arrows shown. The vertices 1 and 4 are in the same orbit; the vertices 5 and 7 are in a different orbit, and the other vertices are in their own orbits.

We use the BLISS library [9] for isomorphism testing and generation of automorphism groups for graphs.

Graphs G_1 and G_2 are isomorphic if there is a relabeling G_2' of the vertices in G_2 such that $G_2' = G_1$. Alternatively, there is some permutation of the rows and columns of G_2's adjacency matrix such that it is the same as G_1's (permutations are restricted: two rows can only be swapped if the corresponding columns are also swapped and if they are of the same color). BLISS provides a method to generate a canonical representation of a graph, such that two graphs have the same canonical representation if and only if they are isomorphic. The theoretical complexity of graph isomorphism is unknown [8], but existing algorithms, while exponential in worst-case running time, work well for small graphs in practice [11].

The automorphism group of a particular graph G is the set of relabelings of G's vertices that leave G unchanged. In other words, if G is an adjacency matrix, the automorphism group is the set of permutation matrices P such that $PG = G$. The generators of the automorphism groups are some subset of the permutations that can be composed to get every member of the group. BLISS provides a method to find the generators of a graph's automorphism group.

Related to the notion of an automorphism group is the idea of vertex orbits. Two vertices v_i, v_j are in the same orbit if there is some member of the automorphism group that maps v_i to v_j. They are in the same k-orbit if there is a member of the automorphism group that changes at most k elements and maps v_i to v_j. Each vertex belongs to exactly one orbit. Orbits are calculated by iterating over the automorphism generators and combining the indices of vertices that map to one another in a disjoint-set data structure [3, Chapter 21]. Each orbit has a canonical vertex which can be selected by choosing the vertex with the lowest number.

BLISS does not support edge coloring. To implement this feature, when graphs are sent to BLISS, each edge is bisected by a vertex with a color corresponding to the edge color.

3.2 Generating Graphs

If the contributions of two isomorphic graphs to a correlator are equal, then evaluating two isomorphic graphs is a duplication of effort. Eliminating isomorphic graphs can reduce the total number of needed computations by several orders of magnitude. In general, a particle with u up quarks and d down quarks has $u!d!$ possible graphs. Even for an alpha particle nucleus, there are 518400 separate graphs. However, each of these is isomorphic to one of only 236 graphs.

The problem of generating an isomorph-free set of graphs with given properties is discussed at length in [12]. A very simple approach to generating a set of isomorph-free graph is presented in Figure 3-2. This algorithm builds up the graph one source vertex at a time, connecting every possible set of sink vertices to the source vertex, keeping a graph only if it is the first graph generated in its isomorphism class.

Although the number of graphs output by the algorithm is relatively small, the number of intermediate graphs is much larger. To see why, consider the number of graphs in $S^{(n_v-1)}$ compared to the number of graphs in $S^{(n_v)}$. Each graph G in S^{n_v} will generate as many graphs in $S^{(n_v-1)}$ as there are orbits in G_{snk}, since graphs in $S^{(n_v-1)}$ can be generated by removing an edge from a source vertex to its set of sink vertices in a graph in $S^{(n_v)}$. This does not precisely determine the number of graphs

```
Data: A starting graph $G$ which has no edges of color $C$ and $n_v$ source
      vertices, which can have a total of $n_e$ edges added
Result: A set of graphs with all edges of color $C$ added
$S^{(0)} := \{G\}$;
for $i \leftarrow 1$ to $n_v$ do
    $S^{(i)} := \{\}$;
    for $G \in S^{(0)}$ do
        for $V \in \{\{v \in G_{snk} \mid \text{degree}(v) < \text{valence}(v)\} \mid |V| = \text{valence}(v_i)\}$ do
            $G' := G$;
            for $v_j \in V$ do
                $G' := G', (v_i, v_j)$;
            end
            $S^{(i)} = S^{(i)} \cup G'$;
        end
    end
    Remove duplicates of graphs in the same isomorphism class in $S^{(i)}$.
end
```

Figure 3-2: A simple algorithm for finding an isomorph-free set. This algorithm is slow because it requires a very large number of comparisons between graphs.

in S^i, but implies that it grows roughly exponentially for $0 < i < \frac{n_v}{2}$ and shrinks roughly exponentially for $\frac{n_v}{2} < i < n_v$.

This exponential growth makes this algorithm very slow, since graph comparisons take many memory accesses. Fortunately, the isomorph-free set can be generated much more efficiently without doing any graph comparisons. To understand how this is possible, consider the reasons that graphs in the same isomorphism class can appear in $S^{(i)}$:

1. Starting from graph G, v_i is connected to two sets that are equivalent under an automorphism relation (see Figure 3-5c).

2. Connecting v_i to set V_1 starting from graph G_1 creates the same graph as connecting v_i to set V_2 starting from graph G_2 (see Figure 3-5a).

The first source of isomorphic graphs can be eliminated by adding sink vertices only if they are the canonical vertex in their orbits before adding the edges. The edges must be added one at a time to allow for multiple identical edges to be added to the same graph.

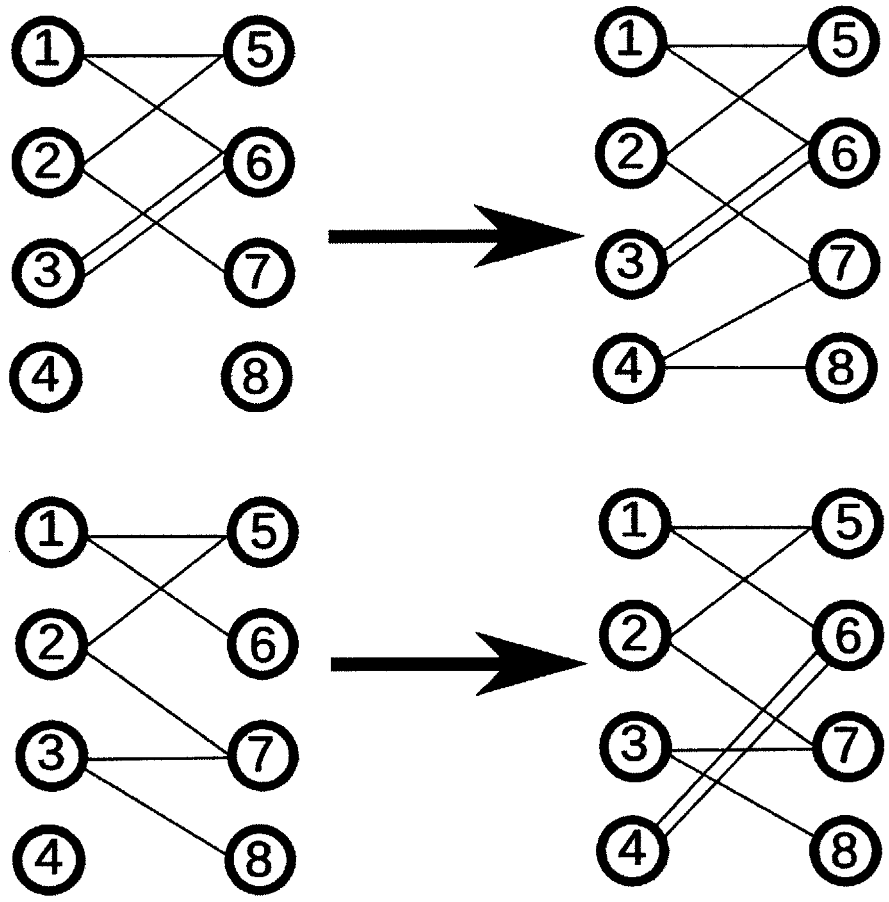


Figure 3-3: Adding different vertices to different graphs can produce two graphs that are isomorphic to each other. Only accepting graphs where the most recently added graph is the lowest numbered orbit would prevent at least one of these graphs from being generated.

```
Data: A starting graph $G$ which has no edges of color $C$ and $n_v$ source
      vertices, which can have a total of $n_e$ edges added
Result: A set of graphs with all edges of color $C$ added
$S^{(0)} := \{G\}$;
for $i \leftarrow 1$ to $n_v$ do
    $S^{(i)} := \{\}$;
    for $G \in S^{(0)}$ do
        for $V \in \{\{v \in G_{snk} \mid \text{degree}(v) < \text{valence}(v)\} \mid |V| = \text{valence}(v_i)\}$ do
            $G' := G$;
            for $v_j \in V$ do
                if $v_j$ is the canonical vertex in its orbit in $G'$ then
                    $G' := G, (v_i, v_j)$;
                else
                    continue;
                end
            end
            if $v_i$ is in the lowest numbered vertex orbit in the canonical labeling
            of $G'$ then
                $S^{(i)} = S^{(i)} \cup G'$;
            end
        end
    end
end
```

Figure 3-4: A fast algorithm for finding an isomorph-free set.

The second source can be eliminated by accepting a newly created graph only if v_i is in the lowest-numbered orbit.

The revised algorithm is presented in Figure 3-4.

In other words, there are four conditions for connecting a sink vertex v_j to the current source vertex v_i:

- C1: v_j must have an available edge to add
- C2: Sink vertices must be added in order - v_j must have a higher index than v_i's other connections.
- C3: v_j must be the canonical vertex in its orbit - no two equivalent edges should be added to the same graph.

- C4: Once a set of v_js is connected, v_i must be in the lowest-numbered vertex orbit in the canonical representation.

It can be shown that this procedure generates a member of every isomorphic group and does not generate any duplicates.

Since we build up the graphs by adding one source vertex at a time, we can inductively prove that every graph is generated at least once; that is, given $S^{(k-1)}$, we can produce each graph in $S^{(k)}$.

Consider a particular graph G in $S^{(k)}$. It has k subgraphs that have $k-1$ attached source vertices. Each of these has a set of edges that can be added back to get G.

Suppose we have every graph with $k-1$ source vertices. To prove that we generate the graph G with k source vertices, consider each of its subgraphs with $k-1$ source vertices. For each of these, there is a set of augmentations that can be added to get G.

Condition C1 does not prevent any of these augmentations from happening, since the vertices we need to augment all have open edges by definition.

Condition C2 means that only one permutation of each set of augmentations gets added to each graph.

Condition C3 prevents two equivalent edges from being added to the graph, but some edge in that orbit will still be added.

C4 allows an augmentation from exactly one subgraph that gives G, since G has some source vertex v_{low} that has the lowest number in the canonical representation. A subgraph of G that has had v_{low} disconnected is the only one that can be augmented to get G.

Next, we show that at most one copy of each graph is generated.

Suppose we have an isomorphism-free set $S^{(k-1)}$ and two isomorphic graphs G and G' are generated in $S^{(k)}$.

Either G and G' were generated from the same subgraph, or they were generated from different subgraphs.

Suppose they were generated from different subgraphs F' and F''. Condition C4 implies that the vertex added to F' must be in the same orbit as the vertex added to

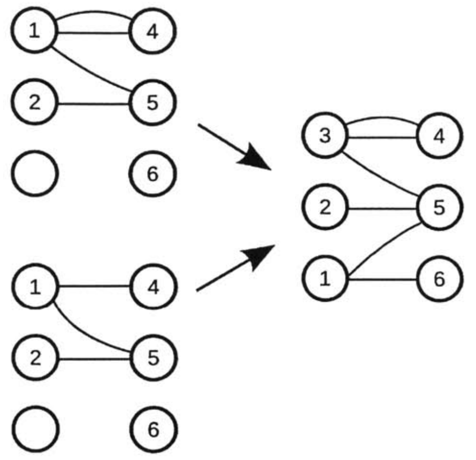
F'', as seen in Figure 3-5a. This means that F' and F'' must be isomorphic, which contradicts the definition of $S^{(k-1)}$, as in Figure 3-5b.

Otherwise, G and G' were generated from the same $(k-1)$-subgraph S. If v_s in G connects to vertices $v_1, v_2, v_3, \ldots$, then v_s in G' must connect to vertices $v'_1, v'_2, v'_3, \ldots$, where v_i is in the same orbit as v'_i. If $v_i \neq v'_i$, only the lower numbered one can be added to S because of C3, as demonstrated in Figure 3-5c. Therefore, all the v_i and v'_i are the same. C2 ensures that each possible set of three vertices is added only once. Therefore, a particular graph S will not be augmented in two different ways to produce isomorphic graphs.

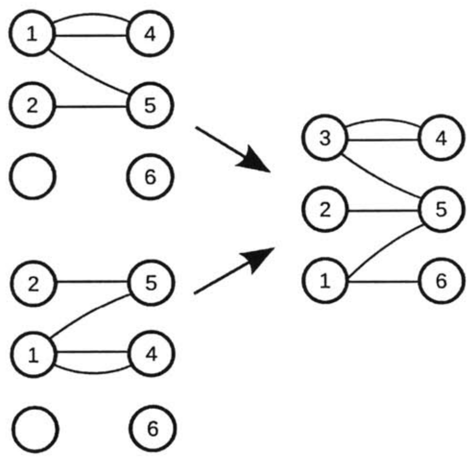
3.3 Application

This approach is useful if two isomorphic graphs have correlators that can be easily calculated from one another. For example, if all the particles were on the same site, two isomorphic graphs would have equal or opposite correlators, depending on the number of exchanges between the graphs. However, for non-trivial wave functions, this leads to the value of any operator being zero due to the Pauli exclusion principle.

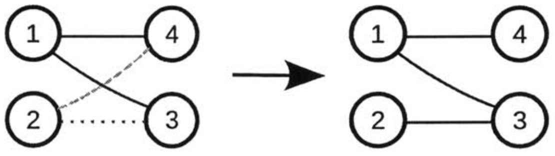
Other applications of this strategy are being investigated. Applying a similar approach to systems of mesons gives a recurrence that allows large systems to be quickly computed by extending smaller systems [5], and it is possible that the tree of graphs generated by the algorithms reveals a similar recurrence.



(a) Either of the graphs on the left could be augmented to get the the graph on the right. However, only the top augmentation has the added vertex as the vertex numbered 1, so the bottom augmentation will not be allowed by condition C4.



(b) Both of the graphs on the left could be augmented to produce the graph on the right. However, since they are isomorphic, the inductive condition means that only one of them will exist.



(c) Adding the dashed green edge or the dotted red edge would produce a graph isomorphic to the one on the right. Only the red edge can be added, because vertices 3 and 4 are in the same orbit, and 3 has a lower labeling. C3 prevents the green edge from being added.

Figure 3-5: An illustration of permitted and forbidden graph augmentations. The numbering of the vertices represents the canonical labeling. Yet-to-be-added vertices on the left are not considered for the labeling.

Bibliography

[1] Silas R Beane, Emmanuel Chang, Saul D Cohen, William Detmold, HW Lin, TC Luu, K Orginos, A Parreno, MJ Savage, A Walker-Loud, et al. Light nuclei and hypernuclei from quantum chromodynamics in the limit of su (3) flavor symmetry. *Physical Review D*, 87(3):034506, 2013.

[2] John M Bennett. Triangular factors of modified matrices. *Numerische Mathematik*, 7(3):217–221, 1965.

[3] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[4] William Detmold and Kostas Orginos. Nuclear correlation functions in lattice qcd. *Physical Review D*, 87(11):114512, 2013.

[5] William Detmold and Martin J Savage. Method to study complex systems of mesons in lattice qcd. *Physical Review D*, 82(1):014511, 2010.

[6] Jiu Ding and Aihui Zhou. Eigenvalues of rank-one updated matrices with some applications. *Applied Mathematics Letters*, 20(12):1223–1226, 2007.

[7] Paola Favati, Mauro Leoncini, and Angeles Martinez. On the robustness of gaussian elimination with partial pivoting. *BIT Numerical Mathematics*, 40(1):62–73, 2000.

[8] Scott Fortin. The graph isomorphism problem. Technical report, Technical Report 96-20, University of Alberta, Edomonton, Alberta, Canada, 1996.

[9] Tommi A Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *ALENEX*, volume 7, pages 135–149. SIAM, 2007.

[10] G Peter Lepage. Lattice qcd for novices. *arXiv preprint hep-lat/0506036*, 2005.

[11] Brendan D McKay. *Practical graph isomorphism.* Department of Computer Science, Vanderbilt University, 1981.

[12] Brendan D McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26(2):306–324, 1998.

[13] Abhay K Parekh. Analysis of a greedy heuristic for finding small dominating sets in graphs. *Information processing letters*, 39(5):237–240, 1991.

[14] Steven J Plimpton and Karen D Devine. Mapreduce in mpi for large-scale graph algorithms. *Parallel Computing*, 37(9):610–632, 2011.

[15] Peter Stange, Andreas Griewank, and Matthias Bollh. On the efficient update of rectangular lu-factorizations subject to low rank modifications. *Electronic Transactions on Numerical Analysis*, 26:161–177, 2007.

[16] Lloyd N Trefethen. Three mysteries of gaussian elimination. *ACM Signum Newsletter*, 20(4):2–5, 1985.