QUN WANG
Reducibility and Flows on Feynman Diagrams
(Under the Direction of ROBERT W. ROBINSON)

A Feynman diagram of order n is a rooted mixed graph on $2n$ vertices consisting of a perfect matching of undirected edges (called V-lines) and a permutation formed by directed edges (call G-lines). A Feynman diagram is irreducible if and only if it is connected and can't be disconnected into two disjoint components by removing 2 G-lines. Three algorithms to test the reducibility of Feynman diagrams were implemented and tested. The three algorithms are called quadratic, pseudolinear and randomized. Conservative flows are constructed in order to test for reducibility in all three algorithms. For a Feynman diagram with order n, the quadratic algorithm takes expected time $O(n^2)$. The pseudolinear algorithm takes expected time $O(n)$ for orders up to 63, but takes expected time $O(n^2)$ for arbitrarily higher orders. The randomized algorithm takes expected time $O(n \log n)$ for diagrams of any order, but is effectively linear for $n \leq 2^{15}$. Testing was done on Feynman diagrams of various orders. Timing results are reported and analyzed. For order less than 23, the quadratic and pseudolinear algorithms run faster than the randomized algorithm because of the overhead of checking false cut-pairs in the latter approach. For larger n, the randomized algorithm becomes much faster than the other two. The pseudolinear algorithm runs faster than the quadratic algorithm due to its smaller constant factor, even for higher order diagrams.

INDEX WORDS: Feynman diagram, conservative flow, reducibility, pseudolinear algorithm, quadratic algorithm, randomized algorithm

REDUCIBILITY AND FLOWS ON FEYNMAN DIAGRAMS

by

QUN WANG

B.S., Wuhan University, China, 1991

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2001

REDUCIBILITY AND FLOWS ON FEYNMAN DIAGRAMS

by

QUN WANG

Approved:

Major Professor: Robert W. Robinson

Committee: Heinz-Bernd Schüttler

E. Rodney Canfield

Electronic Version Approved:
Gordhan L. Patel
Dean of the Graduate School
The University of Georgia
December 2001

ACKNOWLEDGMENTS

The research for this thesis was supported in full by the U.S. National Science Foundation under NSF Grant No. ACI-0081789.

I would like to express my sincere appreciation to the following people:

Dr. Robert W. Robinson, who served as my major professor, for his great guidance and helpful instructions on my academic research and supporting me during my two years' studies at Athens. He spent countless hours in discussion with me. He is always willing to help and is available all the time even on late afternoons. Dr. Robinson guides me through my research step by step and always takes his students' learning needs into consideration. Thank you very much for your guidance, encouragement, and patience.

Dr. Heinz-Bernd Schüttler, for his support and kindness and great advisement on my research and study.

Dr. Canfield for his great guidance on my thesis and his willingness to serve on my committee.

My husband, for his love, continuous support and encouragement.

My parents, parents-in-law, my younger sister and brother-in-law, for their love, support and understanding.

Finally, I dedicate this thesis to my son, Kun Liu, who is the warmest part in my world.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Feynman diagrams were introduced into quantum field theory by Richard Feynman in 1949 [3, p.8]. They are used to calculate rates for electromagnetic and weak interaction particle processes. The diagrams provide a convenient shorthand for the quantum field calculations. They are a code physicists use to talk to one another about their calculations.

In a Feynman diagram each line represents the propagation of a free elementary particle and each vertex represents an interaction of elementary particles. Feynman diagrams provide for the theory of Quantum Electrodynamics (QED) an equally elegant procedure for calculation: imagine a process that can be carried out by electrons and photons, draw a diagram, and then use the diagram to write the mathematical formula for the quantum-mechanical amplitude for that process to occur. Roughly speaking, the diagrams display the flow of electrons and photons during a scattering process [2, p.3].

Feynman diagram expansions are used in quantum physics to express the energy of a system of particles, such as free electrons in a crystal [9]. For this and more general systems of fermions, the graphs of order n can be viewed as a matching on $2n$ vertices (edges in the matching are undirected and called V-lines) along with a permutation of the $2n$ vertices (represented by directed edges called G-lines). Note that a G-line may form a loop (corresponding to a fixed point of the permutation) or join two edges which are matched by a V-line. One of the G-lines is designated as representing the external

momentum. Most Feynman expansions employ only connected diagrams. For the free electron and more general fermion problems it is also helpful to arrange the expansion so that only irreducible diagrams are considered. Interacting fermion problems are of fundamental importance in a wide range of research areas, including fields as diverse as the electronic structure theory of solids, strongly correlated electron physics, quantum chemistry, and the theory of nuclear matter [10, 11, 12].

1.2 BASIC CONCEPTS

***V*-line**: also called a wavy line, or a boson line, which is an undirected edge.

***G*-line:** also called a straight line, or a fermion line, which is a directed edge.

For each vertex in a Feynman diagram, there is one outgoing *G*-line, one incoming *G*-line and one *V*-line.

order of a Feynman diagram: the number of wavy lines, usually denoted by n.

For an order n Feynman diagram, there are exactly $2n$ *G*-lines and n *V*-lines. Hence, there are $3n$ edges in all in any Feynman diagram of order n.

For purposes of computation in this thesis, the vertices of an order n Feynman diagram are named $\{0, 1, \ldots, 2n-1\}$, and the *V*-line labeled i joins the vertices $2i$ and $2i+1$, for $i = 0, \ldots, n-1$. Further, the *G*-lines are labeled $\{0, 1, \ldots, 2n-1\}$ so that the one with label j is incident from vertex j.

reducible Feynman diagram: a Feynman diagram D is reducible if and only if it is disconnected, or it can be disconnected into two disjoint components by cutting two *G*-lines. In the latter case we call these 2 *G*-lines a **cut-pair**. Sometimes this is called **G-reducibility**.

Here the connectivity refers to connectivity of the underlying multigraph (multiple edges are allowed) obtained by considering the *G*-lines to be undirected.

irreducible Feynman diagram: a Feynman diagram which is not reducible.

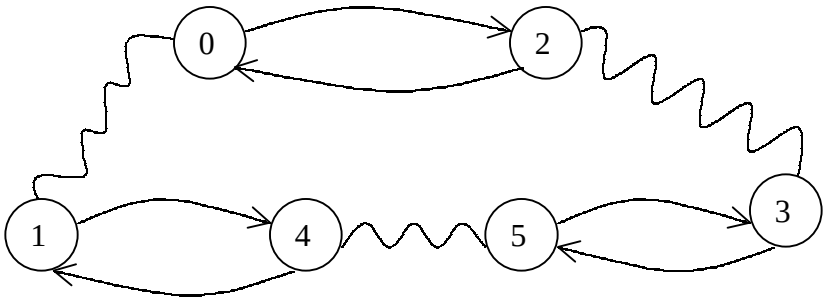

Figure1.1: An example of a Feynman Diagram of order 3

3-edge-connected graph**:** an undirected graph *G* is 3-edge-connected if and only if $G - S$ is connected for any set *S* of edges with $|S| \leq 2$.

contraction of Feynman diagram *D*: the undirected graph which is obtained by contracting all *V*-lines and disregarding the directions of all *G*-lines, denoted by D^*. The relationship between the contraction and the Feynman diagram is that *D* is irreducible if and only if D^* is 3-edge-connected.

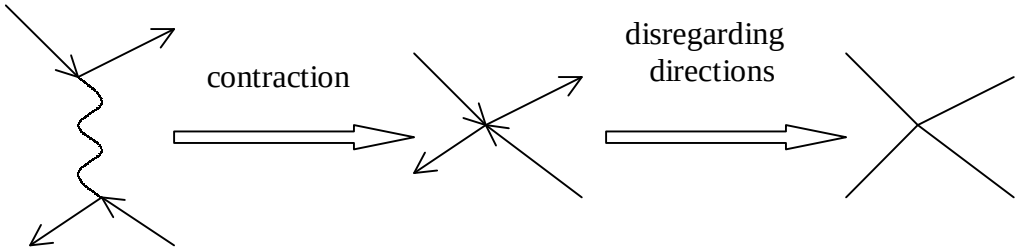

Figure 1.2: An example of contracting a *V*-line

spanning tree: a tree T is said to be a spanning tree of a graph G if T is a subgraph of G and T contains all the vertices of G. On a Feynman diagram, only spanning trees which contain all of the V-lines are of interest.

chord: if T is a spanning tree in a graph G, then a chord is an edge of G which is not in T.

fundamental cycle: a cycle which is formed by adding a chord to a spanning tree. (A connected graph G is a tree if and only if adding an edge between any two vertices in G creates exactly one cycle.)

In an order n Feynman diagram, there are $2n$ vertices and $3n$ edges. Hence $2n$–1 edges are tree edges (n V-lines and n–1 G-lines) and n+1 edges are chords (and are all G-lines).

1.3 OUTLINE OF THE THESIS

Quadratic, pseudolinear and randomized algorithms for determining the reducibility of an arbitrary Feynman diagram are investigated. The thesis is organized as follows:

Chapter 2 describes the motivation for this thesis, and proves two theorems which are the theoretical foundation of our algorithms. Building on the two theorems, a conservative flow will be constructed on a given Feynman diagram, and then used to check for the reducibility of the diagram.

Chapter 3 describes a quadratic algorithm to decide the reducibility of a Feynman diagram. The algorithm applies depth-first-search (DFS) to build a DFS tree which contains all the V-lines. The n+1 G-lines which are the chords are assigned independent current value variables (k_0, …, k_n). The current assignments to the tree edges are linear combinations of the chord variables and can be updated by adding one chord at a time. The check for a Feynman diagram's reducibility is accomplished by simple operations on

an array which records the current for each *G*-line. It is shown in Chapter 2 that a Feynman diagram is reducible if and only if some two of its *G*-lines always carry identical current values. The whole procedure makes just one pass through the diagram and takes expected time $O(n^2)$.

Chapter 4 describes a pseudolinear algorithm to detect the reducibility of Feynman diagrams. It also applies DFS to build a DFS tree, assigns independent currents to the chords and computes the depth-first search index (DFI) according to the order of discovery of the vertices. After the DFS tree is completed, the current for each tree edge is calculated in reverse order of the depth-first search index. Let r denote the size of the array needed to represent the chord bits; for instance, $r = \lceil (n+1)/64 \rceil$ for an array of **unsigned long long int** elements when these represent 64 bit values. Then the algorithm takes expected time $O(rn)$, which is $O(n^2)$ for arbitrarily large n but which behaves in a linear fashion for $n \leq 63$.

Chapter 5 describes a randomized algorithm. Instead of assigning unique independent currents which are powers of 2 to the chords, this algorithm generates 31 bit random currents to the chords. It takes expected time $O(n)$ for order up to 2^{15}, which is far beyond the range of practical application or even feasible testing. For arbitrary n the number of bits assigned for a current value needs to the $\Theta(\log n)$, giving an algorithm with an asymptotic expected time of $O(n \log n)$.

In Chapter 6, the performance of the three algorithms is compared and conclusions are drawn.

CHAPTER 2

CONSERVATIVE FLOWS ON FEYNMAN DIAGRAMS

2.1 MOTIVATIONS FOR THE THESIS

We consider network flows on Feynman diagrams. An **A-flow** is an assignment $\varphi : E \to A$, where A is any abelian group and φ satisfies the conservative law (1) below. We denote the group operation for A by + and refer to it as addition. Our assumptions about $(A, +)$ are that A is nontrivial (contains at least two elements), addition is commutative and associative, there is an additive identity which we call 0, and there are additive inverses given by the – operation. For any vertex v, let $In(v)$ denote the set of edges incident to v (ending at v), and $Out(v)$ the set of edges incident from v (starting at v). For a Feynman diagram, $In(v)$ and $Out(v)$ will each contain one G-line. In addition, one of the two will contain a V-line. The convention adopted for the V-line i is to treat it as if directed from $2i$ to $2i+1$, for $i = 0, \ldots, n-1$.

The **conservation law** at vertex v then takes the form

$$\sum_{e \in In(v)} \varphi(e) = \sum_{e \in Out(v)} \varphi(e) \tag{1}$$

We say that φ is a **conservative flow** (or just a **flow**) if it satisfies (1) at every vertex.

In traditional network terminology our flows are network flows with no source, no sink, and no capacity limitation for the edges.

In this thesis there are three motivations for considering flows on Feynman diagrams. Two arise in the applications of Feynman diagram in quantum field expansions. In both

the real space expansion and the phase space expansion, values must be assigned to the edges so as to satisfy (1) at every vertex. This is part of the process which assigns a weight to the diagram, so in the quantum Monte Carlo calculations it will be repeated frequently. Therefore, the assignment of flow values needs to be as efficient as possible.

The third motivation arises from the need to restrict the diagrams considered in the simulation to those that are irreducible. It will be proved in Theorem 2 that a connected diagram is reducible if and only if there is some pair of *G*-lines which are always assigned the same value in every flow. This fact is the basis for the irreducibility algorithms developed in the thesis. As explained by the physicists participating in the Quantum Monte Carlo Simulation project [4], restricting the expansion to irreducible Feynman diagrams serves to improve the convergence behavior of the simulations.

2.2 DEPTH-FIRST SEARCH

The depth-first search (DFS) algorithm [15] visits the vertices of a Feynman diagram according to the following procedure. Initially, no vertex has been visited and no edge has been explored. Vertex 0 is reserved as the root of the tree. Starting with vertex 0, the highest precedence in the search at each vertex is given to the *V*-line, then to the outgoing *G*-line, and last to the incoming *G*-line. The depth-first index (DFI) for each vertex becomes its canonical name starting with vertex 0. Each of the n *V*-lines will be a tree edge along with some $n-1$ of the *G*-lines. It is easily seen that in a connected diagram the DFI of each vertex is unique, in view of the specification of the root vertex.

The tree generated by DFS will serve as the canonical spanning tree *T*. Each chord along with the unique tree path joining its ends forms a fundamental cycle, and these fundamental cycles form a basis for the circuit space of the diagram [8, p.159]. For each

tree edge *e* we maintain the set of chords encountered for which *e* lies on the fundamental cycle. The algorithms in this thesis implement the chord set maintenance in different ways, but they all make essential use of a parent array which records the parent of each vertex other than 0 in the DFS tree.

2.3 CONSERVATIVE FLOWS ON FEYNMAN DIAGRAMS

For a given Feynman diagram, we consider how to construct conservative flows on it. For an arbitrary Feynman diagram *D* of order *n*, DFS takes time *O(n)* and establishes whether or not *D* is connected. Limiting attention to connected diagrams, the DFS which follows the ordering of edges at each vertex specified in section 2.2 is seen to discover the vertices of *D* in an order which depends only on the starting vertex and not on the labeling of the vertices. For convenience we have adopted the convention that the starting vertex is labeled 0. A representation which employs arbitrary labels would need to specify a root vertex instead. In the physical interpretation of the diagram there is a special *G*-line which corresponds to an external field and which is pictured as two half-edges. In our representation the external field corresponds to the *G*-line which is incident to the root, *i.e.*, to vertex 0.

It will be seen that the irreducibility of a Feynman diagram *D* can be decided by analysis of conservative flows on *D*. A connected diagram *D* will be irreducible if and only if there are no two *G*-lines of *D* which carry the same value in every conservative flow. In order to prove this we start with two lemmas.

Throughout let *A* denote some nontrivial abelian group under operations + and – with additive identity 0. For irreducibility tests *A* will always be Z_2^r, the direct sum of *r* copies of Z_2. For $r = 32$ this is implemented in C or C++ by the **int** data type and bitwise XOR

operator ^ for both + and –. For $r = 64$ the **unsigned long long int** type will serve the purpose, and in general $r = 64d$ can be implemented as an array of d **unsigned long long int** elements. If E is an edge set, an A-flow is just a member of A^E, the set of functions from E to A. These form an abelian group by applying + or – to the values at each edge. The conservative A-flows are the subsets which satisfy equation (1) at every vertex. We show first that the conservative A-flows form an abelian subgroup of all A-flows on $E(D)$.

Lemma 2.1: The conservative A-flows on a Feynman diagram D form an abelian group under + and –.

Proof:

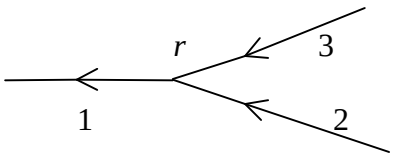

Figure 2.1: Edges at vertex $r = 2i+1$

Assume that there are two conservative A-flows on a diagram D and treat r is a typical vertex of odd index as shown in Figure 2.1. Suppose also that for edges 1, 2, 3 at r the two assignments are $i_1, i_2, i_3,$ and $j_1, j_2, j_3,$ respectively. Then by equation (1), we have:

$$i_1 - i_2 - i_3 = 0. \tag{2}$$

$$j_1 - j_2 - j_3 = 0. \tag{3}$$

Adding (2) and (3) together, we find that the sum of the two flows satisfies the conservative law at vertex v:

$$(i_1 + j_1) - (i_2 + j_2) - (i_3 + j_3) = 0.$$

Similarly, subtracting (3) from (2) gives:

$$(i_1 - j_1) - (i_2 - j_2) - (i_3 - j_3) = 0.$$

In the same way we see that conservation is preserved by the sum and difference of flows at vertices of even index, with one edge in and two out (for flow direction). Thus the sum and difference of conservative flows on a Feynman diagram are also conservative.

Lemma 2.2: Every assignment to the chords of a spanning tree in a Feynman diagram *D* can be extended to a conservative flow on *D*.

Proof: let *Lx* be a fundamental cycle with respect to some spanning tree in a Feynman diagram *D*, and let *e* be the chord in *Lx*.

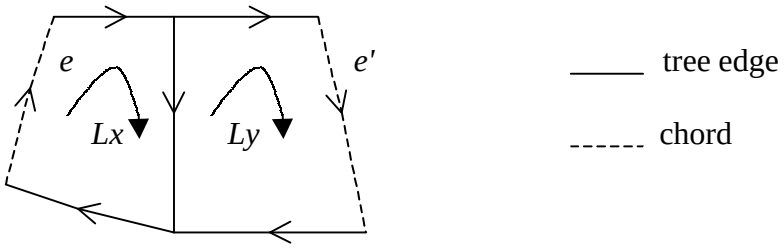

Figure 2.2: An example of an assignment to a chord of a spanning tree being extended to a conservative flow on *D*.

First, assign the chord *e* an independent current *i*. Then for each tree edge *f* in this fundamental cycle assign the current *i* or *–i* depending on whether the direction of *f* agrees with the direction of *Lx* or not. Clearly the assignment currents in this cycle are conservative.

In the same way, we can assign each chord in D an independent current, and extend it to a conservative flow around the chord's fundamental cycle. These flows are called mesh currents in circuit analysis.

By Lemma 2.1, the sum of conservative flows is also conservative. So when we add up all the mesh currents, we actually obtain a conservative flow on the graph.

Every chord is in just one fundamental cycle, so the sum of the mesh currents is a conservative flow which extends the arbitrary flow values assigned to the chords of D.

Theorem 1: Every assignment to the chords of a Feynman diagram D determines a unique conservative flow on D.

Proof: We know by Lemma 2.2 that there is at least one conservative flow on D which extends a particular assignment to the chords. Now suppose that there are two such flows on D, say f and f'.

By Lemma 2.1, since f and f' are both conservative flows, the difference $f - f'$ will also be conservative on D.

And now we need to show those two assignments actually are identical, *i.e.*, $f - f'$ is 0 on every edge of D. Let $N = \{e \in E(D): (f - f')(e) \neq 0\}$. If T is the spanning tree for D, then $N \subseteq E(T)$, since by assumption $f(e) = f'(e)$ if e is a chord.

Since T is acyclic, N is acyclic also.

Suppose there is an edge e' in N. Then e' is in a connected component T' in N. Since $N \subseteq T$, we have $T' \subseteq T$.

It is clear that in an arbitrary non-trivial tree there must be at least 2 end-vertices, that is, vertices with degree = 1. Let v be such an end-vertex of T', and let e'' be the edge in T' at v, say oriented toward v without loss of generality.

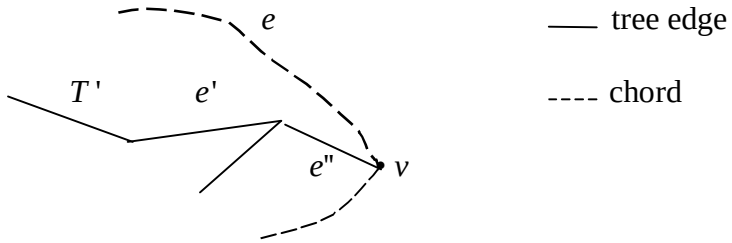

Figure 2.3: An example of part of subtree T' in a Feynman diagram

Now for any other edge, say e, which is not in N and is connected with v, we have $f'(e) = f(e)$ by the definition of N.

Thus $\sum_{e \in In(v)} (f - f')(e) = f(e'') - f'(e'') \neq 0$, while $\sum_{e \in Out(v)} (f - f')(e) = 0$.

Hence $f - f'$ isn't conservative on vertex v. This is a contradiction, since $f - f'$ is conservative on all vertices.

Thus $N = \emptyset$, so $f - f' = 0$ on all edges, *i.e.*, $f = f'$.

This completes the proof of Theorem 1.

Recall that DFS on a given Feynman diagram D generates a canonical spanning tree. The canonical spanning tree will include all n V-lines since those are always explored first in the DFS. The spanning tree has $2n-1$ edges in all, so $n-1$ of them are G-lines. The chords of D are the remaining $n+1$ G-lines which are not in the canonical spanning tree.

The complete topology for a connected Feynman diagram can be specified by two matrixes $\boldsymbol{\sigma_g}$ **(u, v)** and $\boldsymbol{\sigma_v}$ **(x, v)**, where $u \in \{0, \ldots, 2n-1\}$ represents a G-line, $x \in \{0, ..., n-1\}$ represents a V-line and $v \in \{0, \ldots, n\}$ represents a chord. As noted earlier,

we label a *G*-line which originates from vertex *v* as *v*, and label a *V*-line by the even vertex divided by 2. We also adopted the convention that *V*-line *x* carries current q_x from vertex *2x* towards vertex *2x+1* (for $x = 0, ..., n-1$). The resulting $\sigma_g(u, v)$ and $\sigma_v(x, v)$ take on values 0, +1, –1 only. The spanning tree generated by the DFS now uniquely determines the (σ_g, σ_v) pair.

The construction of (σ_g, σ_v) is handled by adaptations of the conservative law. The independent "current" variables ($k_0, ..., k_n$) here are to be assigned only to chords because the spanning tree will include all of the *V*-lines.

The currents k_u and q_x associated with the *G*-lines and *V*-lines respectively [4], are determined by the diagram's topology, based on the current conservation rule at each vertex. In this way the pair (σ_g, σ_v) is constructed so that when currents on the tree edges are calculated by the rules then the resulting flow is conservative on D. In fact, every conservative flow on D is obtained uniquely by (4), which is equivalent to Theorem 1.

$$k_u = \sum_{v=0,...,n} \sigma_g(u,v)\, k_v \qquad \text{and} \qquad q_x = \sum_{v=0,...,n} \sigma_v(x,v)\, k_v \tag{4}$$

2.4 THE REDUCIBILITY OF FEYNMAN DIAGRAMS

As mentioned earlier, the third motivation for considering conservative flows arises from the need to restrict the Feynman diagrams used in the simulation to those that are irreducible.

Since a Feynman diagram *D* is irreducible if and only if it can't be disconnected into two disjoint components by cutting at most 2 *G*-lines, *D* is irreducible if and only if the contraction D^* is 3-edge connected. We now need to prove that: a connected diagram is

reducible if and only if there is some pair of *G*-lines which are always assigned the same value in every conservative flow.

Theorem 2: A connected Feynman diagram *D* is reducible if and only if there are some two G-lines e_1, e_2, such that $\varphi(e_1) = \varphi(e_2)$ for every conservative A-flow on *D*.

Proof:

First we show that if a connected Feynman diagram *D* is reducible then there are some two *G*-lines e_1, e_2, such that $\varphi(e_1) = \varphi(e_2)$ for every conservative A-flow on *D*.

Since *D* is reducible, $D - \{e_1, e_2\}$ is disconnected for some two *G*-lines e_1, e_2. So e_1, e_2 are a cut pair which partitions all vertices in *D* into 2 disjoint sets, say V_1 and V_2.

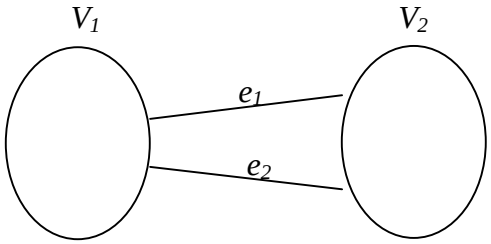

Figure 2.4: A partition of the vertices of *D* into 2 disjoint sets V_1 and V_2 by a cut pair e_1, e_2

Since the *G*-lines of *D* form a permutation, the *G*-line e_1 lies in a directed cycle *C* of *G*-lines. In order for $\{e_1, e_2\}$ to disconnect *D*, they must disconnect *C*, which means that e_2 also lies on *C*. If, say, e_1 is oriented from V_1 to V_2 then e_2 must be oriented from V_2 to V_1 since it is the only other edge between the two sets and $C - e_1$ is a directed path starting in V_2 and ending in V_1.

It is a standard result of network flow theory that the net flow from V_1 to V_2 is the sum of the net flow out of the vertices in V_1 [6, Lemma 4.2.1]. For a conservative flow

the net flow out of any vertex is 0, so the net flow from V_1 to V_2 is 0. In this case, the net flow from V_1 to V_2 is $\varphi(e_1) - \varphi(e_2)$, so $\varphi(e_1) = \varphi(e_2)$ for any conservative flow on D.

Second, we prove that if e_1 and e_2 are different G-lines in diagram D such that $\varphi(e_1) = \varphi(e_2)$ for every conservative A-flow on D, then D must be a reducible diagram. To proceed by contraposition, assume that D is irreducible and that e_1 and e_2 are different G-lines in D.

That means $D - \{e_1, e_2\}$ is connected, so we could build up a new spanning tree, say T, in which e_1 and e_2 are chords.

Let's assign $\varphi(e_1) = 0$, $\varphi(e_2) \neq 0$, and 0 to all other chords. By Lemma 2.2, every assignment to the chords of a spanning tree in D can be extended to a conservative flow on D. Then there exists a conservative flow φ which has these assignments to e_1 and e_2, and for which $\varphi(e_1) \neq \varphi(e_2)$. This contradicts our assumption $\varphi(e_1) = \varphi(e_2)$.

That completes the proof of the Theorem 2.

So checking for a connected Feynman diagram's G-reducibility can be accomplished by simple operations on the $\boldsymbol{\sigma}_\mathbf{g}$ array: from Theorem 2 the diagram is G-reducible if and only if at least two of its G-lines carry identical current, *i.e.*, by equation (4) if and only if for some pair of G-lines (u, u'), with $u, u' \in \{0, \ldots, 2n-1\}$ and $u < u'$, one has $\boldsymbol{\sigma}_\mathbf{g}(\mathbf{u, k}) = \boldsymbol{\sigma}_\mathbf{g}(\mathbf{u', k})$ for $k = 0, \ldots, n$. In practice, since the $\boldsymbol{\sigma}_\mathbf{g}$ array is a $2n \times (n + 1)$ array, a directed search for two identical rows has time complexity $\Theta(n^2)$. In order to improve the performance, we represented each row of $\boldsymbol{\sigma}_\mathbf{g}$ by an array of $r = \lceil (n+1)/64 \rceil$ **unsigned long long int** elements and searched for identical rows by hashing. This can easily be managed, so that the expected number of collisions between rows that are not identical is

$o(1)$. Thus the expected number of hash operations is $O(n)$. Each hash operation takes time $\Theta(r)$, so the total expected time for the process of searching for identical rows by hashing is $O(rn)$. This is $O(n^2)$ for arbitrary n, but behaves linearly for $n \leq 63$.

CHAPTER 3

A QUADRATIC ALGORITHM

3.1 ALGORITHM DESCRIPTION

From the discussion in the chapter 2, we use hashing techniques to check the reducibility for any given Feynman diagram. σ_g, σ_v are also computed for convenience of physicists, who are used to generating conservative flows based on σ_g and σ_v.

As noted before the algorithm is incorporated into a single DFS. Let D be any Feynman diagram. Then D is represented by its associated array **wout[]**, defined as follows: for each vertex i, let **wout[i]** be the vertex to which i is adjacent by its outgoing G-line.

Initialize the matrix pair (σ_g, σ_v) to zero. The chords are indexed 0, ..., n in the order in which they are first explored and each chord is assigned a unique independent power of 2 at the same time. Suppose that the chord of index j, which is also the G-line of index i, is labeled as the G-line (i, **wout[i]**). The column of index j in σ_g and σ_v corresponds to the fundamental cycle of chords j. When chord j is discovered, the non-zero values of $[i, j]$ entry ε of the σ_g and σ_v arrays are determined. The entry ε is set to +1 if **DFI[wout[i]]** < **DFI[i]** and to –1 if **DFI[wout[i]]** > **DFI[i]**. Note that **DFI** is a 1:1 map and for $n > 1$, **wout[i]** = 1 (called a self-loop in D) implies that D is reducible, so the algorithm halts at once in that case.

(1) If the $\varepsilon = -1$, repeated application of the parent function to **wout[i]** until *i* is reached will generate σ_g entry for all of the tree edges on the fundamental cycle of index *j*. The *j* entry of each corresponding row in σ_g is set to 1 if it's oriented away the root by computing the **DFI** function, set to –1 if it's oriented toward the root. And also will generate σ_v entry of all the *V*-lines on the fundamental cycle of index *j*.

(2) If the entry $\varepsilon = +1$, the procedure is varied by starting with *i* and backing down the DFS tree to **wout[i]** and setting σ_g entries to +1 if it's oriented away the root by computing the **DFI** function, set to –1 if it's oriented toward the root. And also will generate σ_v entry of all the *V*-lines on the fundamental cycle of index *j*.

In both cases, for the index *t* of *V*-line which is labeled as (*t*, **parent[t]**), if *t* is even, set σ_v**[t/2, j]** to –1, if *t* is odd, set σ_v**[(t–1)/2, j]** to +1.

The maximum power of 2 which can be represented in a single variable of integer type is implementation dependent. For us it was 2^{63} in the data type of **unsigned long long int**. For an order *n* Feynman diagram there are *n*+1 chords. Thus to compute the reducibility of a Feynman diagram whose order is less than 64, we can directly assign a unique independent current to each chord. But for diagrams of order 64 or more, we need an array of **unsigned long long int** to represent independent currents with values of 2^{64} or more. The current on each tree edge is then calculated as a linear combination of the*n*+1 chord currents.

For example, in order to calculate the reducibility of a diagram of order 1023, we need to use an array of 16 **unsigned long long int** to represent the unique independent

currents of the 1024 chords. Each element in that array represents 64 bits, one bit *per* chord.

3.2 ANALYSIS OF TIME COMPLEXITY

The correctness of the quadratic algorithm is clearly supported by Theorem 1 and Theorem 2. The whole procedure can actually all be done as part of the initial DFS, so that when the search is finished on the canonical **wout[v]**, the σ_g and σ_v arrays are completed. The algorithm makes just one pass through the diagram. When encountering a chord, we need to transverse the fundamental cycle, so in the worst case we may back up the whole spanning tree. So the time for DFS is $O(n^2)$.

In the DFS, we also assign independent current to each chord, in the same time calculated the currents for tree edges which exist in the fundamental cycle formed by that chord. For diagrams of lower order, we can directly assign a unique independent **unsigned long long int** current to each chord. For the order greater than 63 diagrams, we need use an **unsigned long long int** array, say **B[]**, to represent the current value. Since the chord of index m is the j^{th} bit in **B[i]**, where $j = m \bmod 64$ and $i = m / 64$. Since we accomplish the computation for tree edges in the same DFS procedure, even for those higher order diagrams, we know that if the tree path for the chord m contains the tree edge of index v, then v obtains a contributed current by m just 1 bit operation. That means for a Feynman diagram of any order, the assignment computation takes constant time for each chord and tree edge on the chord's fundamental cycle.

As mentioned in the chapter 2, we use hashing techniques to carry out the check for of G-reducibility which takes expected time $O(rn)$ where $r = \lceil (n+1)/64 \rceil$. This behaves linearly for order up to 63, but is $O(n^2)$ for arbitrarily large n.

So the overall time complexity for the quadratic algorithm is $O(n^2)$ in expected time for arbitrary n.

CHAPTER 4

A PSEUDOLINEAR ALGORITHM

4.1 ALGORITHM DESCRIPTION

The algorithm works in the following way:

1. Applying DFS on a given Feynman diagram with order n to generate a unique spanning tree. The chords are assigned to the 0^{th}, 1^{st}, 2^{nd}, . . . , n^{th} power of 2 currents in the order encountered in the DFS. This ensures all chords have been assigned a unique independent current. In the process of DFS, we also recording the depth first index, *i.e.*, **DFI** function and **parent** function when remembers each vertex's parent in the DFS.
2. Calculate inverse depth search index **INDFI** function according to the **DFI** function.
3. According to the I**NDFI** function, we compute the current of each tree edge in inverse depth search index order. Start with the highest depth-first search index which must be an end-vertex of the spanning tree generated in step 1, the current of the tree edge connected with that end-vertex can be calculated as a linear combination of the currents on the two chords connected with that end-vertex. In the same manner, we can compute the current on all the tree edges in the spanning tree, each one as a linear combination of currents on chords or on tree edges already determined.

Since the pseudolinear algorithm uses the same power of 2's assignments to the chords as the quadratic algorithm, so when the order of the Feynman diagram goes

beyond 63 it must also use the strategy of representing independent currents with values of 2^{64} or more with an array of **unsigned long long int**. But for diagrams of order below 64, each chord can just be assigned an **unsigned long long int** current value.

4.2 ANALYSIS OF TIME COMPLEXITY

In general, DFS takes time $O(n+m)$, where n is the number of vertices and m is number of edges in a given graph. For Feynman diagrams, the number of edges is exactly $3n$, so DFS takes time $O(n)$.

For the part to calculate the current for each tree edge on the generated spanning tree, for diagrams of order smaller than 64, we directly assign a unique independent **unsigned long long int** current to each chord. So the currents of tree edges will just be a linear combination of those **unsigned long long int** values. Hence this part also takes time $O(n)$. But for diagrams of order greater than 63, we use an array of **unsigned long long int** to represent the currents. Then the currents on tree edges are calculated by linear combinations of two corresponding **unsigned long long int** array currents. Hence it takes time $O(n^2)$ since the length of each array is $\Theta(n)$.

To test for reducibility, we use hashing techniques as in Chapter 3. This takes expected time $O(rn)$ where $r = \lceil (n+1)/64 \rceil$.

Hence the entire pseudolinear algorithm takes expected time $O(rn)$. It behaves linearly for order less than 64, but is $O(n^2)$ for arbitrarily large n.

CHAPTER 5

A RANDOMIZED ALGORITHM

5.1 ALGORITHM DESCRIPTION

In order to design a more efficient algorithm, we need to make some changes to the pseudolinear algorithm. The objective of the modifications is to ensure that the currents assigned to the chords will not exceed 2^{63}. To avoid the limit number of the power of 2's assignments, we use a 31 bit random number generator - linear congruential generator (LCG) to randomly generate a current to each chord. So each chord can be directly assigned a current. And the assignments of tree edges can be calculated as a linear combination of those chords in the inverse depth search index order.

Since we now assign edges with random numbers, it may happen that two chords have been assigned the same random number. If this happens we should report it as a false cut-pair. Although the probability is very very low, we still need to verify whether a cut-pair is true or not each time when the program reports it. In order to remember the cut-pair, we maintain a **markEdge** array for each *G*-line, initializes to –1 for each edge. If after first calculation, we find a cut-pair, then we mark the corresponding entry in the **markEdge** to 0 respectively. We apply the DFS again on the given diagram, which is now modified by removing the cut-pair from the diagram. If the remaining diagram has become disconnected, the cut-pair is a true, so the given diagram must be reducible. Otherwise the cut-pair is false. If none of the cut-pairs is true then the diagram is irreducible.

5.2 ANALYSIS OF TIME COMPLEXITY

In order to control the impact of DFS verification of reported cut-pairs, we show that the number k of bits in the current group $A = Z_2^k$ should grow at the rate $k = \lceil 2\log_2 n \rceil$.

Let D be a connected Feynman diagram of order n and suppose that $\{e_1, e_2\}$ is a set of two G-lines which are not a cut-pair for D. We know from the proof of Theorem 2 that flow values for e_1, e_2 can be assigned independently, and the rest of the flow is then determined by exactly n–1 additional independent assignments to G-lines. Thus if assignments are made at random from the abelian group A, then the probability that e_1, e_2 are assigned the same value is exactly $1 / |A|$. Since the number of pairs of G-lines is $\Theta(n^2)$, we have by the linearity of expectation that the average number of false cut-pair reports from a random conservative flow is $O(n^2 / |A|)$. When $A = Z_2^k$, the time needed for a + operation on member of A is $\Theta(k)$. Since $|A| = 2^k$, choosing $k = \lceil 2\log_2 n \rceil$ assures that the average number of false cut-pair reports becomes $O(n^2 / n^2) = O(1)$. Each false cut-pair report requires a DFS for verification, taking time $O(n)$. Thus we have $O(n) = o(n \log n)$ total expected time for verifying false cut pairs. This is negligible compared to the $\Theta(n \log n)$ time needed for a DFS with flow assignments. The time for such a DFS is $\Theta(nk)$ in general, because there are $\Theta(n)$ ^ operations, and each ^ operation takes time $\Theta(k)$. In our algorithm k is chosen so that $k = \Theta(\log n)$.

For any disconnected Feynman diagrams, the program will halt as soon as its disconnectedness is detected before finding a cut-pair. So in that case the randomized algorithm takes time $O(n \log n)$.

So for the current assignments the randomized algorithm takes time $O(n \log n)$ if D may not be connected and expected time $\Theta(n \log n)$ if D connected. For orders up to 2^{15},

we can leave k fixed at 31, so the expected time for the randomized algorithm is effectively linear in this range. Note that 2^{15} is much larger than is needed for our applications, and larger than any value used for timing tests.

As in Chapter 3 and 4, hashing is used to test for the potential cut-pair. Since each current is represented in $\Theta(\log n)$, but the time to compute a single hash value will also be $\Theta(\log n)$, so the expected time for the hashing is now $O(n \log n)$ for arbitrary n. For $n \leq 2^{15}$, we just use 31 bits for each current value, so the expected time for hashing becomes linear in this range.

5.3 LINEAR CONGRUENTIAL GENERATOR

As a practical matter, the 31 bits provided by nonnegative values of the **int** data type in *C/C++* are sufficient for the range of the order needed for Monte Carlo simulations ($n \leq 30$) and also for the large numbers used in our timing tests ($n \leq 1023$).

In order to generate near perfect 31 bit random numbers, we adopted Otmar Lendl's package *PRNG* [14] which is a collection of algorithms for generating pseudorandom numbers as a library of C functions, released under the URL (http://www.gnu.org/copyleft/gpl.html).

We use the linear congruential generator (LCG) of *PRNG* utility. The following lists the generator plus a few recommendations on the parameters by *PRNG*.

- Definition:

  $y\_n = a * y\_\{ n-1 \} + b \pmod{p} \quad n>0$

- Name: " lcg(p, a, b, y_0)".
- Properties:
  - Period lengths up p are possible

- Strong linear properties
- The quality of the *PRN* depends very strongly on the choice of the parameters
- prng_is_congruential is TRUE
- prng_can_seed is TRUE. The parameter of prng_seed will be used as *y*_{n–1} in the next call to get_next.
- prng_can_fast_sub and prng_can_fast_con are TRUE. Requesting these subsequence may be show if large skips are involved and b is not 0.

- Parameter selection: If *p* is a power of 2, then *a* mod *4* = 1 and *b* odd will guarantee period length = *p*. If *p* is prime and *b* = 0 then any prime-root modulo *p* as a will guarantee period length *p* – 1. (y_0 ≠ 0)

As recommendation, we use the following parameters

modulo *p*	multiplicator *a*
$2^{31} - 1 = 2147483647$,	950706376 (*b* = 0)

The 31 bit random number generator is working as following:

Define a global integer *prev*, which records the previous random number, initialized to 1.

Call myrand() function each time to randomly generated a 31 bit random number

function int myrand()

long long a=950706376

int p=2147483647

int y

```
    y=(int)((a*prev)%p)
    prev= y
    return y
End function
```

CHAPTER 6

EXPERIMENTAL COMPARISON AND CONCLUSIONS

6.1 COMPARISON OF THE THREE ALGORITHMS

The quadratic algorithm, the pseudolinear algorithm and the randomized algorithm are all implemented separately so that their efficiencies could be compared. For testing purposes, the Feynman diagrams were produced by using a random Feynman diagram generator program. Every new graph is presented as an array of adjacency list rather than an $n \times n$ matrix because an $n \times n$ array would need to allocate contiguous space in main memory for $n \times n$ elements, which is $\Theta(n^2)$. However, with the array of adjacency lists, the space allocation is $\Theta(n)$ for Feynman diagrams.

The difference between the quadratic algorithm and the pseudolinear algorithm is that the first approach finishes all computation in once depth-first search. Each time encountering a chord, we need assign the an independent current to that chord, also assign the same current to all tree edges existing in the fundamental cycle formed by that chord. In order to do that we need back up the spanning tree each time to accomplish the assignments. But the second approach separates the assignment for chords and tree edges into two procedures. First it applies the depth-first search to build up the spanning tree and identifies the chords and tree edges, in the same time it also assigns an independent current to each chord. After DFS, the pseudolinear algorithm calculates the assignment for each tree edges by a linear combination of currents of chords or tree edges which are already determined in inverse depth search index order. So for the lower order diagrams, two approaches should have no much difference in the time complexity. But when the

order increases, the second approach must shows advantages and run faster compared to the first approach. But when the order greater than 63, since by the implementation limitation on the power of 2's assignments, to assign an independent current to the 64^{th} chord or higher, we need to use an array of **unsigned long long int** to represent the unique independent current of each chord. So in that case the pseudolinear algorithm will be actual a quadratic one, but just with a smaller constant factor comparing with the quadratic algorithm.

In fact, the randomized algorithm is a revision of the pseudolinear algorithm. By avoiding using the power of 2's assignments, it uses a 31 bit random number generator - linear congruential generator (LCG) to randomly generate a current to each chord. This modification avoids the use of an array of **unsigned long long int** assignments for the higher order Feynman diagram, whose order greater than 63. But it may happen that two chords obtain a same random number, although the probability is very very impossible. But we still need to verify a cut-pair is true or not when each time it's reported. That's the extra overhead of the third approach comparing with the first two algorithms. So we expect that when the order of Feynman diagram is low, the randomized algorithm will not show any advantage compared to the quadratic and pseudolinear algorithm. But when the order increases, this approach will show its great efficiency.

To test this hypothesis, we have done the time testing on Feynman diagrams of different orders. Table 6.1 shows the average testing times in seconds *per* 10000 diagrams for the same order diagrams by the three algorithms with orders ranging from 2 to 40. We could see that for diagrams of order smaller than 23 the pseudolinear approach doesn't show many advantages compared to the quadratic approach. And the randomized

algorithm runs slower because the overhead of verifying a true cut-pair. When the order greater than 23, the different trends of three algorithms are clearly shown. The quadratic algorithm runs most slowly among the three algorithms, whereas the randomized algorithm runs most fast.

Table 6.1: Average times in seconds *per* 10000 diagrams with order between 2 to 40 for three algorithms

Order	Quadratic Algorithm	Pseudolinear Algorithm	Randomized Linear Algorithm
2	1.31100	1.52500	2.83700
3	1.52400	1.72400	3.80650
4	1.66133	1.86467	3.56067
5	1.92917	2.17917	3.56233
6	2.25500	2.41000	3.85400
7	2.45585	2.59118	4.19883
8	2.68000	2.82534	4.30000
9	2.88902	3.01653	4.60648
10	3.11000	3.25500	4.90000
11	3.39536	3.45953	5.22132
12	3.88400	3.81400	5.43000
13	4.20774	4.08857	5.55446
14	4.36568	4.75302	6.01999
15	4.62250	5.06000	6.25000
16	5.10934	5.02134	6.37067
17	5.54485	5.62419	6.22734
18	6.07206	5.67006	7.46466
19	6.24485	6.06435	7.04891
20	6.66333	6.50660	7.26000
21	7.19965	6.91965	7.48645
22	7.99721	7.48021	7.72561
23	8.44516	7.62856	7.45186
24	8.88800	7.94800	7.79200
25	9.43333	8.39583	7.92913
26	9.85443	8.55863	8.14653
27	10.49856	9.23856	8.09096
28	10.97625	9.47825	8.65185
29	11.69699	10.19869	9.05269

30	12.31500	10.67000	8.78500
31	12.74156	11.18636	9.05176
32	13.72946	11.52530	9.20000
33	14.17967	12.07307	9.09127
34	15.16408	12.42138	9.74957
35	15.65743	13.80823	9.75893
36	16.60268	13.91458	10.27778
36	16.60268	13.91458	10.27778
37	17.39019	14.17739	9.65079
38	18.11385	14.85215	10.02545
39	18.75325	15.29515	10.57525
40	19.51337	16.14667	10.82667

The Figure 6.1 perfectly illustrates the run-time increasing trends for the lower order Feynman diagrams for those three approaches.

We also test the higher order Feynman diagrams. The results of timing experiments in Table 6.2 shows the average times in seconds *per* 10000 diagrams with order between 2 and 1023 for the three algorithms. It indicates that when the order greater than 30, the randomized approach shows its advantages and efficiency compared to the other two approaches. And it also indicates that when the order of given diagrams greater than 63, the pseudolinear algorithm will be a quadratic time complexity, but it has a smaller constant factor comparing to that of the quadratic algorithm. The timing result shows that the average testing time for the quadratic approach, which starts from 1.31100 seconds *per* 10000 diagrams of order 2 up to 3034.90000 seconds for diagrams of order 1023, increases more dramatically than pseudolinear one, which starts from 1.52500 seconds up to 658.64999 seconds. With the order n reached 1023, the randomized algorithm was almost 4.4 times faster than the pseudolinear algorithm, and 20 times faster than the quadratic algorithm.

Table 6.2: Average times in seconds *per* 10000 diagrams with order between 2 and 1023 for three algorithms

Order	Quadratic Algorithm	Pseudolinear Algorithm	Randomized Linear Algorithm
2	1.31100	1.52500	2.83700
10	3.11000	3.25500	4.90000
20	6.66333	6.50660	7.26000
30	12.31500	10.67000	8.78500
40	19.51337	16.14667	10.82667
50	28.02500	22.78330	13.30000
60	38.70000	30.19000	15.09000
70	77.00000	64.16666	15.28330
80	74.16670	69.33340	16.26667
90	78.00000	75.00000	19.95000
100	81.66660	76.66660	18.16660
150	130.83333	115.00000	27.50000
200	180.00030	133.33330	26.66660
250	258.33300	162.50000	41.66670
300	334.16670	195.00000	45.00000
350	454.99970	221.66670	70.83300
400	540.00011	233.33330	62.00000
450	689.99997	270.00000	60.00000
500	800.00030	300.00030	73.33330
550	925.83330	330.00030	91.66666
600	1070.00000	340.00000	90.00000
650	1343.28000	390.00000	97.50000
700	1610.00000	361.66700	72.33330
750	1650.00000	425.00000	112.50000
800	2000.00300	506.66600	120.00000
850	2380.00000	538.33400	127.50000
900	2445.00000	555.00000	150.00000
950	2881.66700	601.66700	158.33400
1000	2866.67000	616.66700	133.33330
1023	3034.90000	658.64999	153.49997

The Figure 6.2 demonstrates the advantages of the randomized algorithm compared to the quadratic and pseudolinear approaches for higher order diagrams. Thus, our asymptotic analysis of the three algorithms is clearly supported by the timing test results.

6.2 CONCLUSION

In this thesis, three algorithms were explored to determine the reducibility of Feynman diagrams. All three algorithms give correct results. When the orders of the diagrams are low, the quadratic algorithm works as well as the pseudolinear algorithm. The randomized algorithm runs more slowly than the other two approaches because of its overhead of checking potential cut-pairs. When the order of diagrams increases, the randomized approach shows its advantages and efficiency.

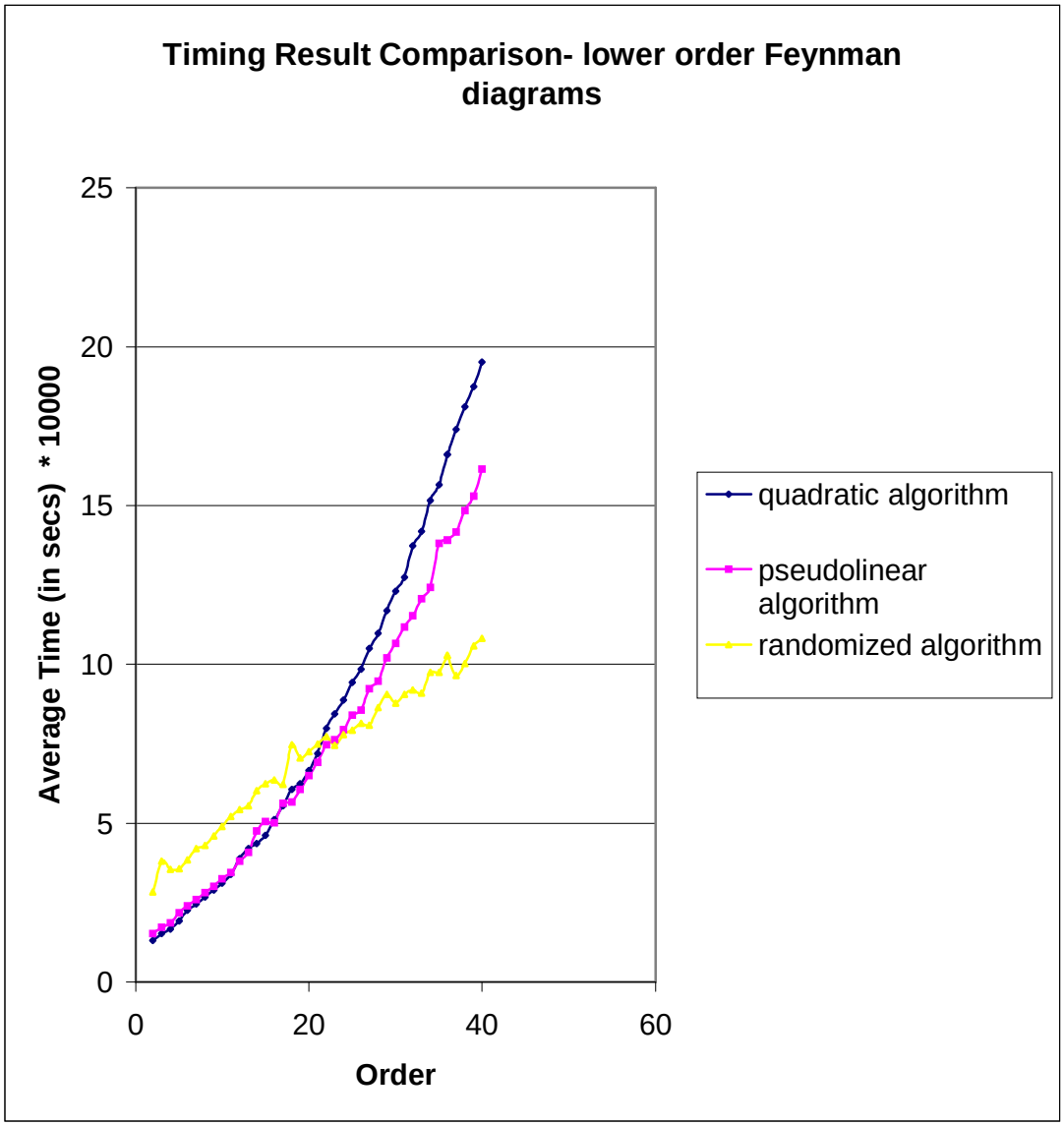

Figure 6.1: Timing comparisons on lower order diagrams for the three algorithms

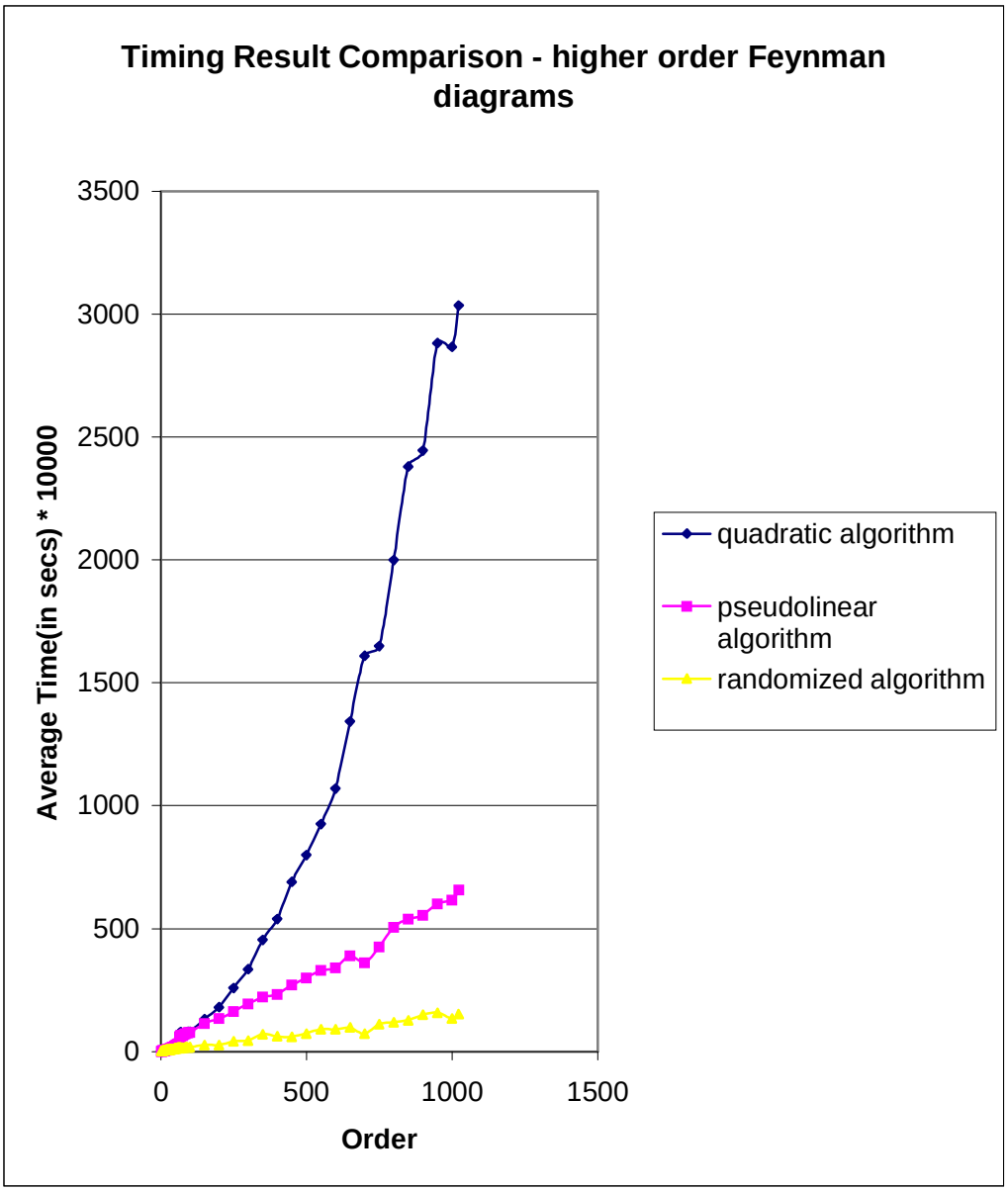

Figure 6.2: Timing comparisons on higher order diagrams

for the three algorithms

REFERENCES

1. Deo, Narsingh, *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall, Englewood Cliffs, NJ, 1974.
2. Peskin, Michael E and Schroeder, Daniel V., *An Introduction to Quantum Field Theory*. Addison-Wesley, Reading, MA, 1995.
3. Nakanishi, Noboru, *Graph Theory and Feynman Integrals*. New York: Gordon and Breach Science Publishers, New York, 1971.
4. Schüttler, H.B., Corcoran, J.N., Lowenthal, D.K. and Robinson, R.W., *Stochastic Summation of High-order Feynman Graph Expansions*. NSF Proposal Number: 0081789 (2000).
5. Harary, Frank, *Graph Theory and Theoretical Physics*, Academic Press, London, 1967.
6. Gould, Ronald, *Graph Theory*, The Benjamin/Cummings, Menlo Park, CA, 1988.
7. Jungnickel, Dieter, *Graphs, Networks and Algorithms*, Springer-Verlag, Berlin, 1994.
8. Gross, Jonathan and Yellen, Jay, *Graph Theory and its Application*, CRC Press, Boca Raton, FL, 1999
9. Robinson, R. W., *Counting irreducible Feynman diagrams,* CATS seminar, 1999.
10. Fulde, P. *Electron Correlations in Molecules and Solids, 3rd ed*., Springer, Berlin, 1995.

11. Maha, G. D., *Many-Particle Physics, 2nd ed*. Plenum, New York, 1990.

12. Negele, J.W. and Orland, H., *Quantum Many-Particle Systems,* Addison Wesley, Redwood City, CA, 1998.

13. http://www2.slac.stanford.edu/vvc/theory/feynman.html

14. Lendl, Otmar and Leydold, Josef, *PRNG- Generating random numbers*, version 3.0, 2000.

15. Tarjan, R. E., *Depth-First Search and Linear Graph Algorithms,* SIAM J. Comput. 1, 146-160, 1972.