Rheinland-Pfälzische Technische Universität
Kaiserslautern-Landau
Fachbereich Mathematik

Massively Parallel Algorithms for the Computation of Feynman Integrals

Ali Traore

Vom Fachbereich Mathematik der Rheinland-Pfälzischen Technischen Universität Kaiserslautern-Landau

zur Verleihung des akademischen Grades

Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.)

genehmigte Dissertation

1. Gutachter: Prof. Dr. Wolfram Decker
2. Gutachter: Prof. Yang Zhang, Ph.D.

Tag der Disputation: 21.08.2025

D 386

"Impossible is just an opinion."

Paulo Coelho

Abstract

In this thesis, we explore the relation between mirror symmetry, algebraic geometry, tropical geometry, and computational methods to address two central challenges at the interface of mathematics and physics: evaluating multi-loop Feynman integrals in high-energy physics and computing Gromov-Witten invariants of elliptic curves. Our work bridges enumerative geometry and quantum field theory through innovative algorithms and parallel computing strategies. We generalize Feynman integral evaluations to include ψ-classes for descendant Gromov-Witten invariants, moving beyond trivalent graphs, and express higher-degree integrals as quasimodular forms (polynomials in Eisenstein series E_2, E_4, and E_6), facilitating number-theoretic analysis.

Our first focus is to compute Gromov-Witten invariants via mirror symmetry, which relates these invariants to Feynman integrals associated with trivalent graphs. We develop an optimized algorithm that significantly outperforms existing methods, efficiently generating the series of these invariants. This algorithm uses tropical geometry as a combinatorial bridge, incorporates graph symmetries (e.g., multiple-edge structures), and employs signature matching to group equivalent Feynman integrals. We also prove a vanishing theorem, identifying conditions under which certain Feynman integrals vanish, reducing computational complexity. The Algorithm is implemented in Singular and OSCAR, the latter uses Julia's just-in-time compilation for faster execution, the algorithm is parallelized using GPI-Space, a high-performance workflow management system developed by Fraunhofer ITWM, resulting in significant speedups for large-scale combinatorial computations.

Our second focus addresses the computational challenge of multi-loop Feynman integrals in quantum field theory, where efficiency is critical due to their increasing complexity. We present a framework to automate their reduction using Integration-by-Parts (IBP) identities and sector decomposition. To exploit the sector structure of integration-by-parts relations of Feynman integrals in high energy physics, we introduce a novel approach: we develop an algorithm which translates the web of sectors into a directed acyclic graph (DAG). This graph is translated into a Petri net, which we then encode in the XPNet format. This format can then be executed by our implementation relying on the Singular/GPI-Space framework. This enables scalable, parallel IBP reductions by distributing tasks across multiple machines. This method significantly enhances the efficiency for high-energy physics computations.

Zusammenfassung

In dieser Arbeit untersuchen wir die Beziehung zwischen Spiegelsymmetrie, algebraischer Geometrie, tropischer Geometrie und computergestützten Methoden, indem wir zwei zentrale Fragestellungen an der Schnittstelle von Mathematik und Physik betrachten: die Auswertung von Multiloop-Feynman-Integralen in der Hochenergiephysik und die Berechnung von Gromov-Witten-Invarianten elliptischer Kurven. Die Arbeit wendet jeweils innovative Algorithmen und Parallelisierungsstrategien auf die Berechnung von Feynmanintegralen an.

Wir verallgemeinern die Auswertung von Feynman-Integralen, um psi-Klassen für descentant Gromov-Witten-Invarianten miteinbeziehen zu können, wobei wir über trivalente Graphen hinaus gehen. Ausserdem drücken wir Integrale höheren Grades als quasimodulare Formen (Polynome in Eisenstein-Reihen E_2, E_4 und E_6) aus, was deren zahlentheoretische Analyse erleichtert.

Unser erster Schwerpunkt liegt auf der Berechnung von Gromov-Witten-Invarianten mittels Spiegelsymmetrie, die diese Invarianten mit Feynman-Integralen trivalenter Graphen in Beziehung setzt. Wir entwickeln einen optimierten Algorithmus, der existierende Methoden deutlich übertrifft und effizient die Reihen dieser Invarianten generiert. Dieser Algorithmus nutzt tropische Geometrie als Brücke zur Kombinatorik, berücksichtigt Graphensymmetrien (z.B. Mehrfachkantenstrukturen) und verwendet Signaturvergleich zur Gruppierung äquivalenter Feynman-Integrale. Wir beweisen zudem einen Verschwindungssatz, der Bedingungen identifiziert, unter denen bestimmte Feynman-Integrale verschwinden. Dies reduziert die Rechenkomplexität.

Implementiert in Singular und OSCAR - letzteres nutzt Julias Just-in-Time-Kompilierung für verbesserte Perforamance - wird der Algorithmus mit GPI-Space parallelisiert, einem Hochleistungs Workflow-Management-System des Fraunhofer ITWM, das für großskalige kombinatorische Probleme eine Beschleunigung um Grössenordnungen erreicht. Ein zentraler Schwerpunkt unserer Arbeit liegt auf den algorithmischen Herausforderungen von *Multiloop-Feynman-Integralen* in der Quantenfeldtheorie. Aufgrund ihrer schnell wachsenden Komplexität ist eine effiziente Berechnung hier von entscheidender Bedeutung. Wir präsentieren einen Rahmen zur Automatisierung ihrer Reduktion mittels Partieller Integration (IBP) Identitäten und Sektorzerlegung. Wir verwenden einen neuen Ansatz, der die Sektorstruktur der IBP-Relationen von Feynman-Integralen in der Hochenergiephysik ausnutzt: Wir entwickeln einen Algorithmus, der das Netzwerk von Sektoren in einen gerichteten azyklischen

Graphen (DAG) übersetzt. Dieser Graph wird in ein Petri-Netz überfuehrt, das im XPNet-Format kodiert ist. Dieses Format kann dann von unserer Implementierung verarbeitet werden, die auf dem Singular/GPI-Space-Framework basiert. Dies ermöglicht skalierbare, parallele IBP-Reduktionen durch Verteilung der Aufgaben auf mehrere Rechner. Diese Methode verbessert die Effizienz fuer Berechnungen in der Hochenergiephysik signifikant.

Preface

Overview

Feynman integrals and their evaluation play a central role at the connection point between mathematics and physics. In this thesis, we are concerned with such interactions which arise on the one hand from the idea of mirror symmetry and its implications for algebraic geometry and, on the other hand, from the use of methods of algebraic geometry and commutative algebra in the study of Feynman integrals in theoretical physics. Mirror symmetry, for example, helps us to compute Gromov-Witten invariants of elliptic curves by using tropical geometry as a bridge between algebraic geometry and Feynman integrals, which is our first central question. Our second question relies on techniques from computational algebraic geometry to find small integration-by-parts (IBP) systems, which can be applied when evaluating Feynman integrals in high energy physics.

We start out by investigating how the concept of mirror symmetry is connecting Gromov-Witten invariants of elliptic curves to integrals associated to Feynman graphs, so-called Feynman integrals. We want to compute the Gromov-Witten invariants by calculating the contributing Feynman integrals. The Gromov-Witten invariants are collected into a generating series, which is then studied with regard to properties like quasi-modularity and homogeneity. To do this, one has to compute a large number of terms in the series, which is only feasible by highly efficient algorithms and parallelization. We have developed a significantly improved algorithm to determine the Gromov-Witten invariants which outperforms the existing ones by several orders of magnitude. This algorithm has been implemented in the computer algebra systems Singular and OSCAR. In OSCAR, we gain additional performance due to utilizing the just-in-time compilation in the programming language Julia. Finally, the algorithm was modified to enable parallel computation, and implemented relying on C++ and the workflow management system for high-performance computing GPI-Space, which is developed by Fraunhofer ITWM. We observe that our approach can be transferred to other problems of

combinatorial nature. We provide examples demonstrating how our algorithm can achieve significant speed-ups compared to sequential processing.

To further optimize the efficiency of our algorithm, we explored various approaches, including exploiting graph symmetries, in particular, multiple-edge structures, and grouping Feynman integrals through signature matching. We formulate and prove vanishing theorems for Gromov-Witten invariants of elliptic curves, establishing conditions for a given branch type under which certain Feynman integrals vanish. This theorem significantly reduces computation time by eliminating integrals that are guaranteed to be zero.

Integration-by-parts reduction is a key tool in the evaluation of Feynman integrals, for example, in high-energy physics. The central idea is that Feynman integrals arising from terms of the numerator are related by integration by parts and can be reduced to so-called master integrals. Algebraic geometry comes in when observing that the integration-by-parts (IBP) identities can be derived from a certain module over a polynomial ring over a rational function field. We develop a parallel approach to compute small systems of IBP identities for Feynman integral reduction. Given a Feynman graph and target integrals, the aim is to express the target integrals as a linear combination of master integrals. The goal is to generate a small-sized IBP system that captures the necessary relations between integrals. Parallelism is applied across the sectors of each layer to solve the reduction problem efficiently. We use GPI-Space in conjunction with the computer algebra system Singular for a massively parallel implementation.

Short Context on Tropical and Algebraic Geometry

The mathematical foundation of this thesis lies at the intersection of algebraic geometry and tropical geometry, two disciplines that unite to tackle deep questions in enumerative geometry, mirror symmetry, and computational mathematics [1, 2]. Algebraic geometry, a field with roots in 19th-century projective geometry (for example, Bernhard Riemann's analytic theory of curves [3] and David Hilbert's foundational work on invariant theory [4]), studies geometric objects called varieties defined by polynomial equations. Those studies range from classical curves, such as the circle, to modern, more general, higher-dimensional schemes formalized by Alexander Grothendieck [5]. One of the central tools is the Chow group, which classifies algebraic cycles modulo rational equivalence and serves as the framework for intersection theory, enabling precise quantification of how varieties intersect [6]. This thesis

uses these tools to explore invariants like Gromov-Witten invariants, which count curves on a variety satisfying constraints such as passing through specified points.

Tropical geometry, by contrast, emerged in the late 20th century through interdisciplinary work in optimization theory (e.g., Bergman's logarithmic limit sets [7]), physics (via amoebas [8]), and combinatorics (e.g., Newton polytopes [9]). Often termed "algebraic geometry over the tropical semiring $(\mathbb{R} \cup \{\infty\}, \oplus, \otimes)$," where $\oplus$ denotes maximum (or minimum) and $\otimes$ denotes classical addition, tropicalization transforms algebraic varieties into polyhedral complexes. Crucially, tropicalization preserves key invariants such as degree, even when geometric intuition falters (e.g., a tropicalized conic retains degree 2 despite its polyhedral structure [1]). This framework simplifies complex enumerative problems, recasting curve counting as combinatorial intersection computations [2]. A landmark result, Mikhalkin's Correspondence Theorem, establishes that counts of tropical curves match their algebraic counterparts under suitable conditions [2].

This research uses the connections between these fields. Tropicalization bridges analysis and combinatorics, translating algebraic objects (e.g., elliptic curves or projective spaces) into computable tropical models. We apply this link to compute invariants like Hurwitz numbers (counting branched covers of algebraic curves, which correspond to Riemann surfaces over $\mathbb{C}$) and Gromov-Witten invariants for elliptic curves. Historically, algebraic geometry evolved from Riemann's complex-analytic foundations to Grothendieck's scheme-theoretic abstractions [5], while tropical geometry matured through applications in mirror symmetry [10] and combinatorial optimization [1]. Together, they offer a unified lens for modern geometric challenges.

Mirror Symmetry and Elliptic Curves

Mirror symmetry, discovered in the context of string theory in the 1980s, reveals a duality between pairs of Calabi–Yau manifolds, denoted X and $X^\vee$. The complex structure of X is linked with the symplectic structure of $X^\vee$. As a result, enumerative invariants on X, for instance, the Gromov–Witten invariants that count holomorphic curves, correspond to period integrals (which can be interpreted in terms of Feynman integrals) on $X^\vee$ [11, 12]. Since its mathematical formalization in the 1990s, mirror symmetry has become a central framework in modern enumerative geometry, connecting algebraic and symplectic perspectives [13, 14]. Elliptic curves, as one-dimensional Calabi–Yau manifolds, serve as fundamental examples due to their simplicity and explicit mirror symmetry structures.

An elliptic curve X, typically defined by a cubic equation in Weierstrass form, has a one-dimensional complex moduli space parametrized by the j-invariant [15]. Its mirror manifold $X^\vee$ is also an elliptic curve, with the mirror map explicitly constructed using modular forms [16]. This setting reveals remarkable connections between Hurwitz numbers, which count branched covers of Riemann surfaces with specific ramification profiles, and Feynman integrals, which are graph-based integrals analogous to quantum amplitudes in physics [16]. Specifically, the Gromov–Witten invariants of X correspond precisely to sums of Feynman integrals over trivalent graphs on the mirror elliptic curve $X^\vee$. Tropical geometry further enriches this relationship by providing a powerful combinatorial framework to facilitate these computations [2, 17].

Subsequent developments have expanded this picture significantly. Dijkgraaf [2] first described the connection between Hurwitz numbers and Feynman integrals. Later, Böhm, Bringmann, Buchholz, and Markwig [17] used tropical geometry to link Hurwitz numbers to tropical invariants and refined Feynman integrals. Böhm, Goldner, and Markwig [18] further generalized these ideas to descendant Gromov–Witten invariants by introducing psi-classes to manage tangency conditions. These relationships are illustrated in **Figure 1**, showing the connections first introduced by Dijkgraaf and refined by tropical geometry.

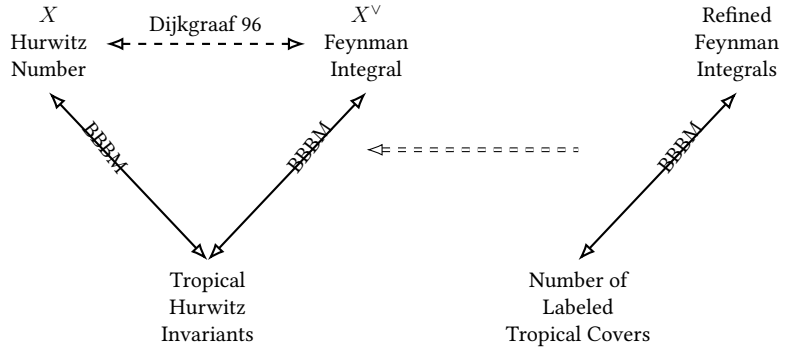


FIGURE 1: Dijkgraaf Correspondence and Tropical Invariants taken from [17]

Feynman Integrals in Mirror Symmetry

The connection between Hurwitz numbers and Feynman integrals is central to mirror symmetry for elliptic curves. Dijkgraaf (1995) established the foundational equivalence:

$$\sum_{d=1}^{\infty} N_{d,g} q^d = \sum_{\Gamma} \frac{1}{|\operatorname{Aut}(\Gamma)|} I_\Gamma(q),$$

where:

- $N_{d,g}$ denotes the Hurwitz number for degree-d covers of genus g,
- Γ ranges over connected trivalent graphs of genus g,
- $I_\Gamma(q)$ is the Feynman integral associated with Γ,
- $|\operatorname{Aut}(\Gamma)|$ is the order of the graph's automorphism group.

This formula bridges algebraic enumerative geometry with physical amplitudes, setting the stage for further exploration.

Tropical Refinement: Böhm et al. (BBBM13) advanced this connection by introducing tropical geometry, proving a chain of equivalences:

$$\text{Hurwitz numbers} \leftrightarrow \text{tropical Hurwitz invariants} \leftrightarrow \text{Feynman integrals}.$$

They showed that the generating series of tropical Hurwitz numbers equals a sum of refined Feynman integrals, which in turn is also equal to the numbers of labeled tropical covers. This implies a direct link between tropical Hurwitz invariants and Feynman integrals, formalized as:

$$\sum_{d=1}^{\infty} N_{d,g}^{\text{trop}} q^d = \sum_{\Gamma} \frac{1}{|\operatorname{Aut}(\Gamma)|} I_\Gamma^{\text{refined}}(q),$$

where $N_{d,g}^{\text{trop}}$ are tropical Hurwitz numbers, and $I_\Gamma^{\text{refined}}(q)$ are refined Feynman integrals. This tropical approach recovers Dijkgraaf's result while providing a combinatorial, graph-by-graph correspondence, reinforcing tropical geometry as a powerful tool in mirror symmetry.

Descendant Invariants (BGM18): Böhm, Goldner, and Markwig (BGM18) generalized this framework to descendant Gromov-Witten invariants of elliptic curves, which extend Hurwitz numbers by incorporating psi-classes (accounting for tangency conditions at marked points). They proved a tropical mirror symmetry theorem, showing that the generating series of descendant Gromov-Witten invariants can be expressed via Feynman integrals:

$$\sum_{d\geq 1} \langle \tau_{k_1}(pt) \cdots \tau_{k_n}(pt) \rangle_{g,n}^{E,d} q^d = \sum_{(\Gamma,\underline{g})} \frac{1}{|\operatorname{Aut}(\Gamma,\underline{g})|} I_{\Gamma,\underline{g}}(q),$$

where:

- $(\langle \tau_{k_1}(pt) \cdots \tau_{k_n}(pt) \rangle_{g,n}^{E,d})$ is the descendant Gromov-Witten invariant for an elliptic curve E involving stable maps of degree d, where the source curves have genus g and there are n marked points,

- $(\Gamma, \underline{g})$ denotes decorated graphs with genus assignments,
- $I_{\Gamma,\underline{g}}(q)$ incorporates contributions from the Weierstrass $\wp$-function and Eisenstein series,
- $|\operatorname{Aut}(\Gamma, \underline{g})|$ accounts for symmetries.

This theorem, a corollary of their Tropical Mirror Symmetry Theorem and Correspondence Theorem, aligns with the Fock space approach in mathematical physics (e.g., Li's work in [19, 20]), offering a practical method to compute these complex invariants tropically.

Extension to $E \times \mathbb{P}^1$: In a subsequent paper (BGM18b), Böhm, Goldner, and Markwig extended their methodology to the generating series of Gromov–Witten invariants of $E \times \mathbb{P}^1$, restricting to point conditions. They proved:

$$\sum_{d_1=1}^{\infty} N_{(d_1,d_2,g)} q^{d_1} = \sum_{\mathcal{P}} \frac{1}{|\operatorname{Aut}(\mathcal{P})|} I_{\mathcal{P}}(q), \tag{1}$$

where:

- $N_{(d_1,d_2,g)}$ counts invariants with bidegree (d_1, d_2) and genus g,
- $\mathcal{P}$ denotes tropical objects (e.g., pearl chains),
- $I_{\mathcal{P}}(q)$ is the associated Feynman integral.

Their method uses intermediate equalities involving generalized tropical curve counts (like leaky degree and curled pearl chains) and Feynman integrals as coefficients of power series, making it more applicable to a wider range of tropical curve counting problems. For stationary descendant invariants (with psi-conditions), they note preliminary correspondences (e.g., [21, 22]), but challenges arise due to vertex contributions from 1-point relative descendant invariants of $\mathbb{P}^1 \times \mathbb{P}^1$, lacking a simple form like the elliptic case's hyperbolic sine expression [23].

Methodology Summary: Their results can be visualized as:

$$\text{Gromov–Witten invariants} \leftrightarrow \text{tropical curve counts} \leftrightarrow \text{Feynman integrals},$$

with Equation (1) as the equality of the leftmost and rightmost terms. For elliptic curves, vertex contributions can be computed, but for $E \times \mathbb{P}^1$, the absence of a compact form for $\mathbb{P}^1 \times \mathbb{P}^1$

invariants limits full generalization to descendants. Their methods, outlined in Construction 2.6 see [24] , also opens the way for future research on tangency conditions with the ∞-section.

Computational Challenges and New Developments

Computing Gromov–Witten invariants and Hurwitz numbers presents significant computational challenges, particularly as the genus or the number of marked points grows. The complexity stems from a combinatorial explosion of possible curve configurations, making direct calculations impractical beyond the simplest cases. Tropical geometry offers a solution by transforming these problems into counts of polyhedral objects, yet even this approach requires sophisticated algorithms to be efficient. Drawing inspiration from recent advancements in Feynman integral computations within quantum field theory, this thesis introduces innovative strategies to tackle these challenges effectively.

Optimized Algorithm for Generating Series

A major contribution of this thesis is an optimized algorithm for calculating the generating series of Hurwitz numbers and Gromov–Witten invariants using tropical mirror symmetry. Building on the works of Böhm et al. (2013), which utilized Laurent series expansions of Feynman propagators, this work introduces **flip signatures** labels that group tropical covers with identical contributions. The algorithm eliminates redundant computations by identifying equivalent covers. Moreover, it avoids explicit series expansions by directly computing the relevant coefficients, thereby enhancing computational efficiency.

This algorithm has been implemented in two systems: Singular and OSCAR. The OSCAR implementation, leveraging Julia's just-in-time (JIT) compilation, achieves remarkable speed improvements. For example, computations for genus $g = 4$, which previously took hours, are now completed in minutes (see Table 2.2 for more details). These advancements make large-scale tropical computations far more practical.

Feynman Integral Reductions

This thesis also draws on cutting-edge developments in physical Feynman integral reductions, notably the NeatIBP 1.0 package by Wu et al. (2024) [25]. Developed for perturbative quantum field theory, NeatIBP creates concise integration-by-parts (IBP) identities for Feynman integrals via the use of syzygy and module intersection methods. Unlike the traditional Laporta algorithm, which often produces unwieldy IBP systems, NeatIBP manages propagator degrees to produce smaller, easier-to-handle systems. Using tools like Mathematica, Singular, and SpaSM, it parallelizes computations across Feynman integral sectors, boosting efficiency for multi-loop integrals essential to experiments like those at the Large Hadron Collider.

Inspired by NeatIBP, this thesis adapts syzygy-based methods to optimize tropical computations. We use NeatIBP's approach to generating Petri nets from web sector structures, it turns out to be efficient on computing Gromov–Witten invariants and Hurwitz numbers.

Parallel Computing

Parallel computing plays a crucial role in addressing the computational requirements of both Feynman integrals and tropical calculations. This thesis employs **Petri nets**, bipartite graphs that model resources and operations within the GPI-Space framework, a Petri net-based workflow management system developed by the HPC group at Fraunhofer ITWM. This framework allows the parallelization of algorithmic tasks across multiple nodes, making it ideal for large-scale problems. For IBP reductions, which simplify complex integrals into master integrals, the thesis uses web sector structures and directed acyclic graphs (DAGs) to optimize multi-loop computations. This parallelization strategy significantly reduces computation times.

Tackling Combinatorial Complexity

The combinatorial complexity of these problems is driven by the size of the partition set, denoted partition(n, d), which has a cardinality of $\binom{n+d-1}{d}$. Here, n represents the number of edges, and d the degree. As n and d increase, this set grows rapidly, necessitating parallelization. The thesis rewrites a sequential algorithm for computing Feynman integrals over branch types $\mathbf{a} = (a_1, \ldots, a_n)$ with $d = \sum a_i \geq 1$, adapting it for parallel execution within GPI-Space.

A key improvement is splitting partition(n, d) into d subsets, each corresponding to a specific degree. This allows simultaneous processing of multiple elements $\mathbf{a}$ across these subsets.

Additionally, for each $\mathbf{a}$, the algorithm identifies pairs of flip signatures that yield identical Feynman integrals, cutting redundant calculations in half.

Theoretical and Algorithmic Advances

The thesis proves a theorem establishing conditions under which Feynman integrals vanish for certain branch types. This generalizes a result from BBBM13, which demonstrated that Feynman integrals are zero for graphs containing a bridge, providing a broader framework for simplifying computations.

Another contribution is an algorithm for computing the quasimodularity of Feynman integrals, implemented using the Flint library. By leveraging number-theoretic properties, this algorithm further streamlines calculations.

Finally, in the last chapter, the thesis generates a Petri net from a web sector structure to compute Feynman integrals in the physical IBP context. This parallelization strategy, based on web sectors, optimizes the reduction of integrals into master integrals, echoing the efficiency gains achieved in tropical computations.

All implementations can be found online:
GromovWitten implemented in Julia:
`https://github.com/singular-gpispace/GromovWitten`
GromovWitten implemented in C++ with GPI-Space:
`https://github.com/singular-gpispace/gspc-gromovwitten`
Feynman IBP implemented in Singular and FLINT:
`https://github.com/singular-gpispace/gspc-feynman`

This research was supported by Project B5 of SFB-TRR 195. Gefördert durch die Deutsche Forschungsgemeinschaft (DFG) - Projektnummer 286237555 - TRR 195 [Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Project- ID 286237555 - TRR 195]. Furthermore, I acknowledge the financial support provided by the German Academic Exchange Service (DAAD) through the Mathematics in Industry and Commerce (MIC) program.

List of Contributions

This thesis comprises six chapters, progressing from foundational concepts to original contributions:

- **Chapter 1: Kontsevich space** This short chapter, authored solely by the thesis author, establishes some basic the mathematical groundwork [1, 26, 27].
- **Chapter 2: Algorithms for Gromov–Witten Invariants of Elliptic Curves** is the result of a joint project with J. Böhm, F. Dastur, A. Hoffmann, H. Markwig. This chapter has been published in the book The Computer Algebra System OSCAR [28].
- **Chapter 3: Towards Parallel Algorithms for Gromov-Witten Invariants of Elliptic Curves** is solely done by the thesis author, it has been published in [29].
- **Chapter 4: Parallel computation of Gromov-Witten invariants** is the result of a joint project with J. Böhm. A slightly different version of this chapter has been published [29].
- **Chapter 5: Vanishing Theorems for Gromov-Witten Invariants** is the result of a joint project with J. Böhm. We intend to publish this chapter separately.
- **Chapter 6: Automated Transformation of the Sector Structure of Feynman Integrals into Petri Nets, and Their High-Performance Execution using GPI-Space** is the result of a joint project with J. Böhm. We intend to publish this chapter separately.

Contents

List of Contributions

List of Figures

List of Tables

List of Algorithms

List of Contributions

Acknowledgements

Remerciements

Je tiens tout d'abord à exprimer ma profonde gratitude à mon directeur de thèse, le Professeur Wolfram Decker, pour ses conseils avisés et ses critiques constructives qui ont considérablement enrichi ce travail. Mes remerciements vont également au Docteur Janko Boehm pour sa disponibilité constante et son accompagnement tout au long de mes recherches.

Je remercie sincèrement le DAAD (Service allemand d'échanges universitaires) et le SFB TRR 195 (Centre de recherche collaborative) pour leur soutien financier qui a rendu cette recherche possible.

Je souhaite témoigner toute ma reconnaissance au Docteur Hans Schönemann pour son assistance précieuse dans le développement des aspects numériques de ce travail, ainsi qu'au Docteur Falk Triebsch et à Jessica Borsche pour leur soutien constant durant l'ensemble de ce doctorat.

Je remercie chaleureusement Jonas Frank pour son aide déterminante dans la rédaction du résumé en allemand, ainsi que Diego et Assoumane pour leurs relectures attentives et leurs suggestions pertinentes. Je tiens aussi à remercier le Docteur Mirko Rahn pour son aide précieuse dans la compréhension de GPI-Space.

Une mention toute particulière revient à Monsieur Siaka Coulibaly, qui m'a accueilli comme son propre enfant et soutenu financièrement jusqu'à l'obtention de mon master. Qu'il trouve ici l'expression de ma gratitude infinie.

Je ne saurais oublier ma mère, mes frères et sœurs, ainsi que tous ceux — trop nombreux pour être cités — qui m'ont apporté leur soutien tout au long de ce parcours.

Je tiens également à adresser des remerciements spéciaux à Santosh Gnawali, compagnon de bureau, pour nos innombrables discussions, aussi enrichissantes sur le plan académique que sur le plan humain.

Enfin, cette thèse n'aurait pu voir le jour sans le soutien indéfectible de ma famille et plus particulièrement de mon épouse, Hahoua Coulibaly, qui m'a accompagné avec patience et dévouement tout au long de ce doctorat, me soutenant inconditionnellement et préservant cet équilibre si précieux entre vie académique et vie personnelle.

Acknowledgements

First and foremost, I would like to express my deep gratitude to my thesis advisor, Professor Wolfram Decker, for his guidance and constructive feedback, which have greatly enriched this work. My sincere thanks also go to Dr. Janko Boehm for his constant availability and support, and for supervising me throughout these years.

I gratefully acknowledge the financial support provided by the DAAD (German Academic Exchange Service) and the SFB TRR 195, which made this research possible.

I am deeply grateful to Dr. Hans Schönemann for his valuable assistance with the Singular implementations in this work, as well as to Dr. Falk Triebsch and Jessica Borsche for their continuous support throughout the course of this PhD.

I warmly thank Jonas Frank for his decisive help in writing the German summary, as well as Diego and Assoumane for their careful reading and thoughtful suggestions. I would also like to thank Dr. Mirko Rahn for his valuable help in understanding GPI-Space.

A special mention goes to Mr. Siaka Coulibaly, who welcomed me as his own child and supported me financially until I obtained my master's degree. May he find here the expression of my deepest gratitude.

I cannot forget my mother, my brothers and sisters, and all those (too numerous to name) who supported me throughout this journey.

I also wish to extend special thanks to Santosh Gnawali, my office companion and the source of countless discussions that were as enriching academically as they were personally.

Finally, this thesis would not have been possible without the unwavering support of my family, and in particular my wife, Hahoua Coulibaly, who accompanied me with patience and devotion throughout this doctoral journey, supporting me unconditionally and maintaining that precious balance between academic and personal life.

To my family

Chapter 1

Kontsevich space

We define here the moduli space of stable maps that we will use to define the Gromov-Witten invariants. This section is based on the following papers: [26] and [27].

Definition 1.0.1. An *n-pointed smooth curve*

$$\mathcal{C} = (C, x_1, \ldots, x_n)$$

is a projective smooth curve C equipped with a choice of n distinct points $x_1, \ldots, x_n \in C$, called the *marks*. A smooth n-pointed curve is called rational if it has genus 0, and elliptic if it has genus 1. An isomorphism between two n-pointed curves

$$\phi : (C, x_1, \ldots, x_n) \xrightarrow{\sim} (C', x'_1, \ldots, x'_n)$$

is an isomorphism $\phi : C \to C'$ that respects the marks, meaning $\phi(x_i) = x'_i$ for all i. For any $g \geq 0$, the set of all smooth n-pointed curves of genus g, up to isomorphism, is denoted by $\mathcal{M}_{g,n}$. This space is called the *moduli space of smooth n-pointed curves of genus g*.

Definition 1.0.2. A *stable n-pointed curve of genus g* is a tuple

$$\mathcal{C} = (C, x_1, \ldots, x_n)$$

where C is a curve with only simple nodes as singularities, and x_i are distinct smooth points on C.

An isomorphism between two stable n-pointed curves

$$\phi : (C, x_1, \ldots, x_n) \xrightarrow{\sim} (C', x'_1, \ldots, x'_n)$$

is an isomorphism $\phi : C \to C'$ that respects the marks, meaning $\phi(x_i) = x_i'$ for all i. The stability condition is equivalent to the condition that the automorphism group of $\mathcal{C}$ is finite. The moduli space of stable n-pointed curves of genus g is denoted by $\overline{\mathcal{M}}_{g,n}$.

A point $p \in C$ is called *special* if it is either a marked point or a singularity of C.

To define Gromov-Witten invariants, we first introduce a more general moduli space known as the *moduli space of stable maps*.

Definition 1.0.3. Let X be an algebraic scheme and let β be an element of the first Chow group $A_1(X)$. A *stable map* of degree β from a genus-g curve to X with n marked points is a morphism $f : C \to X$, where:

- C is a connected, projective curve with at worst nodal singularities,
- $x_1, \ldots, x_n \in C$ are distinct, nonsingular marked points,

such that:

1. $f_*[C] = \beta$,
2. the automorphism group of the map f is finite.

We denote by $\overline{M}_{g,n}(X, \beta)$, the Kontsevich moduli space of stable maps of n-pointed curves of degree β. The existence of $\overline{M}_{g,n}(X, \beta)$ has been proved in the case when X is a projective, algebraic scheme over $\mathbb{C}$.

Theorem 1.0.4. *[30] There exists a projective, coarse moduli space $\overline{M}_{g,n}(X, \beta)$.*

When $X = \mathbb{P}^n$, we have $A_1(X) = \mathbb{Z}$. We will then write $\overline{M}_{g,n}(X, d)$ instead of $\overline{M}_{g,n}(X, dH')$, where H' is the class of a line (so $\beta = dH'$). The space $\overline{M}_{g,n}(X, \beta)$ is endowed with n evaluation maps:

$$\begin{aligned} \mathrm{ev}_i : \overline{M}_{g,n}(X, \beta) &\to X, \\ (C, x_1, \ldots, x_n, f) &\mapsto f(x_i). \end{aligned}$$

To compute the Gromov-Witten invariant, we consider the pullback of the evaluation map

$$\mathrm{ev}_i^* : A^*(X) \to A^*(\overline{M}_{g,n}(X, \beta)),$$

where for a (closed) subvariety $V_i \subset X$, we have

$$[V_i] \mapsto \mathrm{ev}_i^*[V_i].$$

The counting strategy involves computing the product (intersection)

$$\mathrm{ev}_1^*[V_1] \cdots \mathrm{ev}_n^*[V_n],$$

with appropriate choices of the subvarieties V_i and the number n so that the product forms a cycle of dimension 0. The challenge lies in controlling the dimensions at the boundary of the moduli space, where components may exceed the expected dimension. To address this, Behrend and Fantechi [31] introduced the notion of the virtual fundamental class, which replaces $[\overline{M}_{g,n}(X,\beta)]$ in such cases.

If $\dim[\overline{M}_{g,n}(X,\beta)] = m$, then

$$[\overline{M}_{g,n}(X,\beta)]^{\mathrm{virt}} \in A_m(\overline{M}_{g,n}(X,\beta))$$

in the case where X is convex.

Consider the line bundles associated to the n markings. For each marking x_i, let

$$\pi : \mathbb{L}_i \to \overline{M}_{g,n}(X,\beta)$$

be the cotangent line bundle, with fiber given by the cotangent space $T^*_{x_i}(C)$ at x_i. The first Chern class of $\mathbb{L}_i$, denoted by

$$\psi_i = c_1(\mathbb{L}_i),$$

is called a psi class.

Chapter 2

Algorithms for Gromov–Witten Invariants of Elliptic Curves

This chapter presents an enhanced algorithm for exploring mirror symmetry in elliptic curves through the correspondence of algebraic and tropical geometry, focusing on Gromov–Witten invariants of elliptic curves and, in particular, Hurwitz numbers. We present a new highly efficient algorithm for computing generating series for these numbers. We have implemented the algorithm both using Singular and OSCAR. The implementations significantly outperform the current method provided in Singular. The OSCAR implementation, benefiting in particular from just-in-time compilation, again outperforms the implementation of the new algorithm in Singular by far. This advancement in computing the Gromov–Witten invariants facilitates a study of number theoretic and geometric properties of the generating series, including quasi-modularity and homogeneity.

2.1 Introduction

Mirror symmetry is a deep duality relation motivated by physics. It relates invariants of a manifold and its "mirror manifold" in a way allowing an exchange of methods. Tropical methods find a way into mirror symmetry most prominently in the famous Gross-Siebert program [32].

Here, we present the special case of elliptic curves, for which mirror symmetry is best understood [16]. The tropical side of mirror symmetry for elliptic curves was discovered within the Collaborative Research Center SFB-TRR 195 [17, 18, 24], which is also responsible for the development of OSCAR.

One consequence of mirror symmetry for elliptic curves is the equality of the generating function of Hurwitz numbers (resp. more generally, of descendant Gromov–Witten invariants) to certain Feynman integrals which are complex analytic path integrals whose construction is governed by combinatorics.

Hurwitz numbers are traditional enumerative invariants that count covers of Riemann surfaces satisfying fixed ramification data. Their definition goes back to Hurwitz himself and was used to study the irreducibility of the famous moduli space of smooth genus g curves, a fundamental object in algebraic geometry. In modern mathematics, Hurwitz numbers provide fruitful interactions between several mathematical areas such as geometry, topology, representation theory, combinatorics and mathematical physics [33].

Tropical geometry can be viewed as a degenerate version of algebraic geometry. It has been used successfully to solve enumerative problems in algebraic geometry [2]. In particular, by counting *tropical covers* (see Definition 2.2.5), we can determine Hurwitz numbers via a so-called *correspondence theorem* [17, 34]. It allows to translate any computations we perform involving tropical covers to traditional Hurwitz theory.

In this chapter, we describe how one can use tropical mirror symmetry for elliptic curves in order to compute generating series for tropical Hurwitz numbers of an elliptic curve and their generalizations (up to fixed order) by means of computing Feynman integrals. We discuss explicit algorithms and implementations to handle this task. An algorithm computing Hurwitz numbers via Feynman integrals has been introduced in [17]. It works by computing coefficients in the Laurent series expansion of the propagator of the Feynman integral. Based on this algorithm, we develop in this chapter a significantly improved version which shows a better performance when applied to current research problems. This algorithm works by assigning to each cover a so-called *flip signature* to each cover. Covers with the same signature lead to the same integral. Further improvements are achieved by direct computation of the coefficients of the Laurent series, avoiding explicit expansion altogether. We have created two structurally identical implementations of the algorithm, one using the computer algebra system Singular and one using OSCAR. Both implementations outperform the previous algorithm of [17] in its implementation in Singular significantly, which demonstrates the structural improvement in the algorithm. The OSCAR implementation again by far outperforms the implementation of the new algorithm in Singular (which partially relies on the interpreted Singular language). We attribute this performance improvement to the just-in-time (JIT) compilation features, which turn at run-time easy-to-comprehend Julia code into code which executes almost as fast machine code, but also to superior arithmetic in OSCAR. This demonstrates the potential of the OSCAR platform.

2.2 Mirror Symmetry for Elliptic Curves

2.2.1 Hurwitz Numbers

Hurwitz numbers count branched covers of non-singular curves with a given ramification profile over fixed points. Here, we consider covers of elliptic curves. Hurwitz numbers are topological invariants, in particular they do not depend on the position of the branch points. Moreover, since all complex elliptic curves are homeomorphic to the real torus, numbers of covers of an elliptic curve do not depend on the choice of the base curve. We thus fix an arbitrary complex elliptic curve $\mathcal{E}$.

Let $\mathcal{C}$ be a non-singular curve of genus g and let $\phi : \mathcal{C} \to \mathcal{E}$ be a cover. We denote by d the degree of ϕ, i.e., the number of preimages of a generic point in $\mathcal{E}$. For our purpose, it is sufficient to consider covers which are *simply ramified*, that is, over any branch point exactly two sheets of the map come together and all others stay separate. In other words, the *ramification profile* (that is, the partition of the degree indicating the multiplicities of the inverse images of a branch point) of a simple branch point is $(2, 1, \dots, 1)$. It follows from the Riemann–Hurwitz formula (see e.g. [6], Corollary IV.2.4) that a simply ramified cover of $\mathcal{E}$ has exactly $2g-2$ branch points. Two covers $\varphi : \mathcal{C} \to \mathcal{E}$ and $\varphi' : \mathcal{C}' \to \mathcal{E}$ are isomorphic if there exists an isomorphism of curves $\phi : \mathcal{C} \to \mathcal{C}'$ such that $\varphi' \circ \phi = \varphi$.

Definition 2.2.1 (Hurwitz numbers)**.** Fix $2g-2$ points $p_1, \dots, p_{2g-2}$ in $\mathcal{E}$. We define the *Hurwitz number* $N_{d,g}$ to be the weighted number of (isomorphism classes of) simply ramified covers $\phi : \mathcal{C} \to \mathcal{E}$ of degree d, where $\mathcal{C}$ is a connected curve of genus g, and the branch points of ϕ are the points p_i for $i = 1, \dots, 2g-2$. We count each such cover ϕ with weight $|\operatorname{Aut}(\phi)|$.

Remark 2.2.2. In this definition, we choose the convention to fix a marking of the branch points p_i. In the literature, this is not always the case, leading to a factor of $(2g-2)!$ when compared to our definition.

Definition 2.2.3. We package the Hurwitz numbers of Definition 2.2.1 into a generating series as follows:

$$F_g(q) := \sum_{d=1}^{\infty} N_{d,g} q^{2d}.$$

2.2.2 Tropical Hurwitz Numbers

Definition 2.2.4 (Tropical curves)**.** A (generic) *tropical curve* C (without ends) is a connected, finite, trivalent, metric graph. Its *genus* is given by its first Betti number. The *combinatorial*

type of a curve is its homeomorphism class, that is, the underlying graph without the lengths of edges.

A tropical elliptic curve consists of one edge forming a circle of a certain length. We will fix a tropical elliptic curve E (for example, one having length 1). Moreover, to fix notation, in the following we denote by C a tropical curve of genus g and combinatorial type Γ.

Definition 2.2.5 (Tropical covers)**.** A map $\pi : C \to E$ is a *tropical cover* of E if it is continuous, non-constant, integer affine on each edge and respects the *balancing condition* at every vertex $P \in C$:

For an edge e of C denote by w_e the *weight of* e, that is, the absolute value of the slope of $\pi_{|e}$. Consider a (small) open neighbourhood U of $p = \pi(P)$. Then U combinatorially consists of p together with two rays r_1 and r_2, left and right of p. Let V be the connected component of $\pi^{-1}(U)$ which contains P. Then V consists of P together with three adjacent rays. We say that π is balanced at P if

$$\sum_{R \text{ maps to } r_1} w_R = \sum_{R \text{ maps to } r_2} w_R,$$

where R runs over the three rays adjacent to P and rays inherit their weights from the corresponding edges.

The *degree* of a cover is the weighted number of preimages of a generic point: For all $p \in E$ not having any vertex of C as preimage we have

$$d = \sum_{P \in C : \pi(P) = p} w_{e_P},$$

where e_P is the edge of C containing P. The images of the vertices of C are called the *branch points* of π.

Example 2.2.6. *A tropical cover of degree* 4 *with a genus* 2 *source curve is depicted in Figure 2.1 (taken from [17]). The red numbers close to the vertex* P *are the weights of the corresponding edges; the black numbers denote the lengths. The cover is balanced at* P *since there is an edge of weight* 3 *leaving in one direction and an edge of weight* 2 *plus an edge of weight* 1 *leaving in the opposite direction.*

We can see that the length of an edge of C is determined by its weight and the length of its image. We will therefore not specify edge lengths in the following.

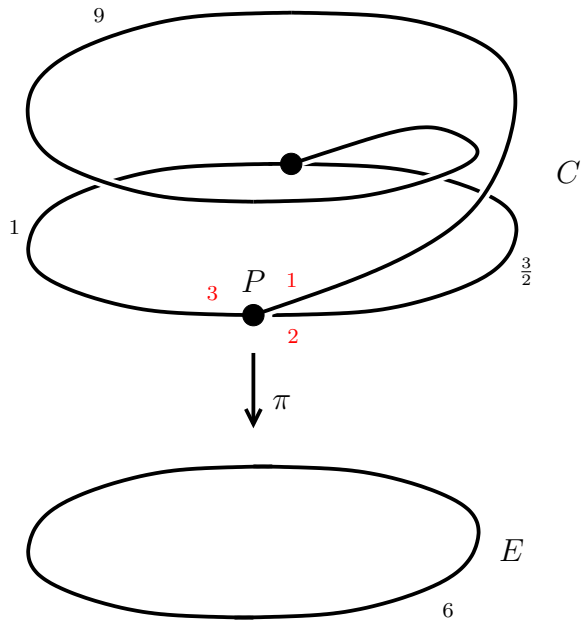


FIGURE 2.1: A tropical cover of degree 4 with genus 2 source curve.

Definition 2.2.7 (Isomorphisms of curves and covers)**.** An *isomorphism of tropical curves* is an isometry of metric graphs. Two covers $\pi : C \to E$ and $\pi' : C' \to E$ are isomorphic if there is an isomorphism of curves $\phi : C \to C'$ such that $\pi' \circ \phi = \pi$.

As usual when counting tropical objects we have to weight them with a certain multiplicity.

Definition 2.2.8 (Multiplicities)**.** The multiplicity of a cover $\pi : C \to E$ is defined to be

$$\mathrm{mult}(\pi) := \frac{1}{|\operatorname{Aut}(\pi)|} \prod_e w_e,$$

where the product goes over all edges e of C and $\operatorname{Aut}(\pi)$ is the automorphism group of π.

Definition 2.2.9 (Tropical Hurwitz numbers)**.** Fix branch points $p_1, \ldots, p_{2g-2}$ in the tropical, elliptic curve E. The *tropical Hurwitz number* $N_{d,g}^{\mathrm{trop}}$ is the weighted number of isomorphism classes of degree d covers $\pi : C \to E$ having their branch points at the p_i, where C is a curve of genus g:

$$N_{d,g}^{\mathrm{trop}} := \sum_{\pi : C \to E} \mathrm{mult}(\pi).$$

Theorem 2.2.10 (Correspondence Theorem)**.** *The algebraic and tropical Hurwitz numbers of simply ramified covers of an elliptic curve coincide (see Definition 2.2.1 and 2.2.9), i.e., we have*

$$N_{d,g}^{\mathrm{trop}} = N_{d,g}.$$

For a proof, see the extension of the result of [35] in [17].

2.2.3 Feynman Integrals

Definition 2.2.11 (The propagator)**.** We define the *propagator*

$$P(z,q) := \frac{1}{4\pi^2}\wp(z,q) + \frac{1}{12}E_2(q^2)$$

in terms of the Weierstraß-$\wp$-function and the Eisenstein series

$$E_2(q) := 1 - 24\sum_{d=1}^{\infty}\sigma(d)q^d.$$

Here, $\sigma = \sigma_1$ denotes the sum-of-divisors function $\sigma(d) = \sum_{m|d} m$.

The variable q above should be considered as a coordinate of the moduli space of elliptic curves and the variable z is the complex coordinate of a fixed elliptic curve. (More precisely, $q = e^{i\pi\tau}$, where $\tau \in \mathbb{C}$ is the parameter in the upper half plane in the well-known definition of the Weierstraß-$\wp$-function.)

Definition 2.2.12 (Feynman graphs and integrals)**.** A *Feynman graph* Γ of genus g is a trivalent connected graph of genus g. For a Feynman graph, we fix throughout a reference labeling $x_1, \ldots, x_{2g-2}$ of the $2g-2$ trivalent vertices and $q_1, \ldots, q_{3g-3}$ of the edges of Γ, respectively.

For an edge q_k of Γ connecting the vertices x_i and x_j, we define a function

$$P_k := P(z_i - z_j, q),$$

where P denotes the propagator of Definition 2.2.11. (The function P_k is invariant under reordering of z_i and z_j.) Pick a total ordering Ω of the vertices and starting points of the form $iy_1, \ldots, iy_{2g-2}$ in the complex plane, where the y_j are pairwise distinct small real numbers. We define integration paths $\gamma_1, \ldots, \gamma_{2g-2}$ by

$$\gamma_j : [0,1] \to \mathbb{C} : t \mapsto iy_j + t,$$

such that the order of the real coordinates y_j of the starting points of the paths equals Ω. We then define the integral

$$I_{\Gamma,\Omega}(q) := \int_{z_j \in \gamma_j} \prod_{k=1}^{3g-3} (-P_k)\,. \tag{2.1}$$

Finally, we define

$$I_\Gamma(q) = \sum_{\Omega} I_{\Gamma,\Omega}(q),$$

where the sum runs over all $(2g-2)!$ orders of the vertices.

The following relates Hurwitz numbers and Feynman integrals and can be viewed as a consequence of mirror symmetry (see Theorem 3 of [16]):

Theorem 2.2.13 (Mirror symmetry for elliptic curves). *Let $g > 1$. For the definition of the invariants, see Definitions 2.2.1 and 2.2.12. We have*

$$F_g(q) = \sum_{d=1}^{\infty} N_{d,g} q^{2d} = \sum_{\Gamma} I_\Gamma(q) \cdot \frac{1}{|\operatorname{Aut}(\Gamma)|},$$

where $\operatorname{Aut}(\Gamma)$ denotes the automorphism group of Γ and the sum goes over all trivalent graphs Γ of genus g.

Theorem 2.2.14. *After coordinate change, the propagator $P(x,q)$ of Definition 2.2.11 with $x = e^{i\pi z}$ equals*

$$P(x,q) = -\frac{x^2}{(x^2-1)^2} - \sum_{n=1}^{\infty} \left(\sum_{d|n} d \left(x^{2d} + x^{-2d} \right) \right) q^{2n}.$$

For a proof, see [17, Theorem 2.22].

To better relate Feynman integrals with counts of tropical covers, we label the sources of our tropical covers just like a Feynman graph and consider labeled tropical covers. This way, we can also refine the notion of degree by considering the degree of each edge of the source curve. Also, we can add information about the edge in the definition of the Feynman integral in Equation 2.1 by making the propagator associated to edge k dependent on a formal variable q_k, see Definition 2.2.25 below.

Definition 2.2.15. We fix a base point p_0 in E. For a tuple $\underline{a} = (a_1, \ldots, a_{3g-3})$ of non-negative integers, we define $N^{\mathrm{trop}}_{\underline{a},g}$ to be the weighted number of labeled tropical covers $\hat{\pi} : C \to E$ of degree $\sum_{i=1}^{3g-3} a_i$ where C has genus g, such that $\hat{\pi}$ has its branch points at the prescribed positions and satisfying the condition

$$\#(\hat{\pi}^{-1}(p_0) \cap q_i) \cdot w_i = a_i$$

for all $i = 1, \ldots, 3g-3$.

Theorem 2.2.16 (Refined version of tropical mirror symmetry). *Let $g > 1$. We have*

$$F_g(q_1, \ldots, q_{3g-3}) = \sum_{\underline{a}} N^{\mathrm{trop}}_{\underline{a},g} q^{2d} = \sum_{\Gamma} I_\Gamma(q_1, \ldots, q_{3g-3}),$$

that is, the coefficient of the monomial $q^{2\cdot a}$ in $I_\Gamma(q_1, \ldots, q_{3g-3})$ equals $N^{\mathrm{trop}}_{\underline{a},\Gamma}$.

Lemma 2.2.17. *Fix a Feynman graph Γ, an order Ω, and a tuple $(q_1, \ldots, q_{3g-3})$ as in Definition 14. We express the coefficient of $q^{2\cdot\underline{a}}$ in $I_{\Gamma,\Omega}(q_1, \ldots, q_{3g-3})$. Assume that $a_k = 0$, and that the edge q_k connects the two vertices x_{k_1} and x_{k_2}. Choose the order of labeling of x_{k_1} and x_{k_2} such that Ω implies $\left|\frac{x_{k_1}}{x_{k_2}}\right| < 1$ for the starting points on the integration paths. Then the coefficient of $q^{2\cdot\underline{a}}$ equals the constant term of the series*

$$\prod_{k|a_k=0}\left(\sum_{w_k=1}^{\infty} w_k\left(\frac{x_{k_1}}{x_{k_2}}\right)^{2w_k}\right)\cdot\prod_{k|a_k\neq 0}\left(\sum_{w_k|a_k} w_k\left(\left(\frac{x_{k_1}}{x_{k_2}}\right)^{2w_k}+\left(\frac{x_{k_2}}{x_{k_1}}\right)^{2w_k}\right)\right). \tag{2.2}$$

For a series

$$F(x_1, \ldots, x_n) = \sum_{\substack{1\leq j\leq n\\ a_j\geq 0}} \alpha(a_1, \ldots, a_n) x_1^{a_1}\cdots x_n^{a_n} \tag{2.3}$$

we write

$$\operatorname{coeff}_{[x_1^{a_1},\ldots,x_n^{a_n}]}(F) = \alpha(a_1, \ldots, a_n). \tag{2.4}$$

Lemma 2.2.18. *Fix a Feynman graph Γ and an order Ω as in Definition 2.2.12 and write*

$$P_{\Gamma,\Omega} = \prod_{k=1}^{3g-3}\left(-P\left(\frac{x_{k_1}}{x_{k_2}}, q\right)\right).$$

Then we have

$$I_{\Gamma,\Omega}(q) = \operatorname{coeff}_{[x_1^0,\ldots,x_{2g-2}^0]}(P_{\Gamma,\Omega}).$$

2.2.4 Descendant Gromov–Witten Invariants

Theorem 2.2.13 can be generalized to descendant Gromov–Witten invariants.
Following Okounkov–Pandharipande [23], these are related to covers with more complicated ramification profiles.

A *stable map* of degree d from a curve of genus g to $\mathcal{E}$ with n markings is a map $f : \mathcal{C} \to \mathcal{E}$, where $\mathcal{C}$ is a connected projective curve with at worst nodal singularities, and with n distinct nonsingular marked points $x_1, \ldots, x_n \in \mathcal{C}$, such that $f_*([\mathcal{C}]) = d[\mathcal{E}]$ and f has a finite group of automorphisms. The moduli space of stable maps, denoted $\overline{\mathcal{M}}_{g,n}(\mathcal{E}, d)$, is a proper Deligne-Mumford stack of virtual dimension $2g - 2 + n$ [31, 36]. The i-th evaluation morphism is the map $\mathrm{ev}_i : \overline{\mathcal{M}}_{g,n}(\mathcal{E}, d) \to \mathcal{E}$ that sends a point $[\mathcal{C}, x_1, \ldots, x_n, f]$ to $f(x_i) \in \mathcal{E}$. The i-th cotangent line bundle $\mathbb{L}_i \to \overline{\mathcal{M}}_{g,n}(\mathcal{E}, d)$ is defined by a canonical identification of its fiber

over a moduli point $(\mathcal{C}, x_1, \ldots, x_n, f)$ with the cotangent space $T^*_{x_i}(\mathcal{C})$. The first Chern class of the cotangent line bundle is called a *psi class* ($\psi_i = c_1(\mathbb{L}_i)$).

Definition 2.2.19. Fix g, n, d and let $k_1, \ldots, k_n$ be non-negative integers with

$$k_1 + \ldots + k_n = 2g - 2.$$

The *stationary descendant Gromov–Witten invariant* $\langle \tau_{k_1}(pt) \ldots \tau_{k_n}(pt) \rangle^{\mathcal{E},d}_{g,n}$ is defined by

$$\langle \tau_{k_1}(pt) \ldots \tau_{k_n}(pt) \rangle^{\mathcal{E},d}_{g,n} = \int_{[\overline{\mathcal{M}}_{g,n}(\mathcal{E},d)]^{\mathrm{vir}}} \prod_{i=1}^{n} ev_i^*(pt)\psi_i^{k_i}, \tag{2.5}$$

where pt denotes the class of a point in $\mathcal{E}$.

In order to define tropical multiplicities, we also need to discuss relative descendant Gromov–Witten invariants of $\mathbb{P}^1$. They are constructed using moduli spaces of *relative stable maps* $\overline{\mathcal{M}}_{g,n}(\mathbb{P}^1, \mu, \nu, d)$, where part of the data specified are the ramification profiles μ and ν which we fix over 0 resp. $\infty \in \mathbb{P}^1$. The preimages of 0 and ∞ are marked. A detailed discussion of spaces of relative stable maps to $\mathbb{P}^1$ and their boundary is not necessary for our purpose, we refer to [37]. We write

$$\langle \mu | \tau_{k_1}(pt) \ldots \tau_{k_n}(pt) | \nu \rangle^{\mathbb{P}^1,d}_{g,n} = \int_{[\overline{\mathcal{M}}_{g,n}(\mathbb{P}^1,\mu,\nu,d)]^{\mathrm{vir}}} \prod_{i=1}^{n} \mathrm{ev}_i^*(pt)\psi_i^{k_i}. \tag{2.6}$$

Tropically, adding descendants amounts to allowing vertices of higher genus and higher valency. For this purpose, we first generalize the definition of abstract tropical curve:

Definition 2.2.20. An *abstract tropical curve* is a connected metric graph Γ, such that edges leading to leaves (called *ends*) have infinite length, together with a genus function $g : \Gamma \to \mathbb{Z}_{\geq 0}$ with finite support. Locally around a point p, Γ is homeomorphic to a star with r half rays. The number r is called the *valence* of the point p and is denoted by $\mathrm{val}(p)$. The vertex set of Γ consists of the nonzero points of the genus function and the points of valence different from 2. The vertices of valence greater than 1 are called *inner vertices*. Besides *edges*, we introduce the notion of *flags* of Γ. A flag is a pair (V, e) of a vertex V and an edge e incident to it. ($V \in \partial e$). Edges that are not ends are required to have finite length and are referred to as *bounded* or *internal* edges.

A *marked tropical curve* is a tropical curve whose leaves are labeled. An isomorphism of a tropical curve is an isometry respecting the leaf markings and the genus function. The *genus*

of a tropical curve Γ is given by

$$g(\Gamma) = h_1(\Gamma) + \sum_{p \in \Gamma} g(p).$$

The *combinatorial type* is the equivalence class of tropical curves obtained by identifying any two tropical curves which differ only by edge lengths.

Definition 2.2.5 introducing tropical covers easily extends to allow such more general source curves.

Definition 2.2.21 (Psi- and point conditions)**.** We say that a tropical cover $\pi : \Gamma_1 \to \Gamma_2$ with a marked end i *satisfies a psi-condition* with power k at i, if the vertex V adjacent to the marked end i has valency $k + 3 - 2g(V)$. We say $\pi : \Gamma_1 \to \Gamma_2$ *satisfies the point conditions* $p_1, \ldots, p_n \in \Gamma_2$ if $\pi(i) = p_i$ for all $i = 1, \ldots, n$.

Fix g, n, d and let $k_1, \ldots, k_n$ be non-negative integers with

$$k_1 + \ldots + k_n = 2g - 2.$$

Let $\pi : \Gamma \to E$ be a tropical cover such that Γ is of genus g and has n marked ends. Fix n distinct points $p_1, \ldots, p_n \in E$. Assume that at the marked end i, a psi-condition with power k_i is satisfied, and that the point conditions are satisfied. The marked ends must be adjacent to different vertices, since they satisfy different point conditions. It follows from an Euler characteristic argument incorporating the valencies imposed by the psi-conditions that Γ has exactly n vertices, each adjacent to one marked end.

Locally at the marked end i, the cover sends the vertex to an interval consisting of two flags f and f'. We define the *local vertex multiplicity* $\mathrm{mult}_i(\pi)$ to be a one-point relative descendant Gromov–Witten invariant:

$$\mathrm{mult}_i(\pi) = \langle \mu_f | \tau_{k_i}(pt) | \mu_{f'} \rangle_{g_i,1}^{\mathbb{P}^1, d_i}, \tag{2.7}$$

where g_i denotes the genus of the vertex adjacent to the marked end i, d_i its local degree, and μ_f resp. $\mu_{f'}$ the ramification profiles above the two flags of the image interval.

We define the multiplicity of π to be

$$\frac{1}{|\operatorname{Aut}(\pi)|} \cdot \prod_i \mathrm{mult}_i(\pi) \cdot \prod_e \omega(e). \tag{2.8}$$

Note that all ends of a tropical cover of E are contracted ends, with image points the points p_i which we fix as conditions in E.

Definition 2.2.22 (Tropical stationary descendant Gromov–Witten invariant of E)**.** For g, n, d, $k_1, \ldots, k_n$ as above, define the *tropical stationary descendant Gromov–Witten invariant*

$$\langle \tau_{k_1}(pt) \ldots \tau_{k_n}(pt) \rangle_{g,n}^{\mathcal{E},d,\mathrm{trop}}$$

to be the weighted count of tropical genus g degree d covers of E with n marked points satisfying point and psi-conditions as above, each counted with its multiplicity as defined in (2.8).

For descendant invariants, a correspondence theorem holds as well:

Theorem 2.2.23 (Correspondence Theorem for descendants)**.** *A stationary descendant Gromov-Witten invariant of $\mathcal{E}$ coincides with its tropical counterpart:*

$$\langle \tau_{k_1}(pt) \ldots \tau_{k_n}(pt) \rangle_{g,n}^{\mathcal{E},d} = \langle \tau_{k_1}(pt) \ldots \tau_{k_n}(pt) \rangle_{g,n}^{E,d,\mathrm{trop}}.$$

For a proof, see [38, Theorem 3.2.1].

Generating series of descendant invariants can also be expressed in terms of Feynman integrals. However to accommodate this setting, we as well need to extend our notion of Feynman integrals.

Definition 2.2.24 (The propagator and the $\mathcal{S}$-function)**.** We define the *propagator* as a (formal) series in x and q,

$$P(x,q) = \sum_{w=1}^{\infty} w \cdot x^w + \sum_{a=1}^{\infty} \left(\sum_{w|a} w \left(x^w + x^{-w} \right) \right) q^a$$

and the $\mathcal{S}$-function as series in z,

$$\mathcal{S}(z) = \frac{\sinh(z/2)}{z/2}.$$

We also introduce a further formal power series in q, which should be viewed as the propagator for loop edges:

$$P^{\mathrm{loop}}(q) = \sum_{a=1}^{\infty} \left(\sum_{w|a} w \right) q^a.$$

Analogous to the way we generalized our notion of abstract tropical curves which can serve as sources of our tropical covers, we have to generalize the notion of Feynman graphs. We allow any graph Γ without ends with n vertices which are labeled $x_1, \dots, x_n$ and with labeled edges $q_1, \dots, q_r$. By convention, we assume that $q_1, \dots, q_s$ are loop edges and $q_{s+1}, \dots, q_r$ are non-loop edges.

Definition 2.2.25 (Feynman integrals)**.** Let Γ be a Feynman graph. Let Ω be an order of the n vertices of Γ. For $k > s$, denote the vertices adjacent to the (non-loop) edge q_k by x_{k_1} and x_{k_2}, where we assume $x_{k_1} < x_{k_2}$ in Ω. We define the *Feynman integral* for Γ and Ω to be

$$I_{\Gamma,\Omega}(q) = \mathrm{Coef}_{[x_1^0 \dots x_n^0]} \prod_{k=1}^{s} P^{\mathrm{loop}}(q) \cdot \prod_{k=s+1}^{r} P\Big(\frac{x_{k_1}}{x_{k_2}}, q\Big).$$

$$I_{\Gamma,\Omega}(q_1, \dots, q_r) = \mathrm{Coef}_{[x_1^0 \dots x_n^0]} \prod_{k=1}^{s} P^{\mathrm{loop}}(q_k) \prod_{k=s+1}^{r} P\Big(\frac{x_{k_1}}{x_{k_2}}, q_k\Big).$$

Finally, we set

$$I_\Gamma(q) = \sum_{\Omega} I_{\Gamma,\Omega}(q),$$

where the sum goes over all $n!$ orders of the vertices of Γ.

If we assume $|x| < 1$ to express the q-constant coefficient of the (non-loop) propagator (that is, the first sum appearing in the propagator series in Definition 2.2.11) as the rational function $\frac{x^2}{(x^2-1)^2}$ (using geometric series expansion), we can view the series from which we take the $x_1^0 \dots x_n^0$-coefficient in the Feynman integral above as a function on a Cartesian product of elliptic curves. The Feynman integral then becomes a path integral in complex analysis as above. Note that using the change of coordinates $x = e^{i\pi u}$ the (non-loop) propagator has the form of Definition 2.2.11.

Definition 2.2.26 (Feynman integrals with vertex contributions)**.** Let Γ be a Feynman graph, and equip it with an additional genus function $\underline{g}$ associating a non-negative integer g_i to every vertex x_i. Let Ω be an order of the n vertices of Γ. We adapt our notion of propagators from Definitions 2.2.24 and 2.2.25 to include vertex contributions: for non-loop edges, we set

$$\begin{aligned}\tilde{P}(\frac{x_{k_1}}{x_{k_2}}, q) = &\sum_{w=1}^{\infty} \mathcal{S}(wz_{k_1})\mathcal{S}(wz_{k_2}) \cdot w \cdot \left(\frac{x_{k_1}}{x_{k_2}}\right)^w \\ &+ \sum_{a=1}^{\infty} \left(\sum_{w|a} \mathcal{S}(wz_{k_1})\mathcal{S}(wz_{k_2}) \cdot w \cdot \left(\left(\frac{x_{k_1}}{x_{k_2}}\right)^w + \left(\frac{x_{k_2}}{x_{k_1}}\right)^w \right) \right) \cdot q^a.\end{aligned}$$

For loop-edges connecting the vertex x_{k_1} to itself, we set

$$\tilde{P}^{\text{loop}}(q) = \sum_{a=1}^{\infty} \left(\sum_{w|a} \mathcal{S}(wz_{k_1})^2 \cdot w \right) q^a.$$

The new variables z_{k_i} are introduced for each vertex in order to take care of the genus contribution.

We define the *Feynman integral with vertex contributions* for Γ, $\underline{g}$ and Ω to be

$$I_{\Gamma,\underline{g},\Omega}(q) = \text{Coef}_{[z_1^{2g_1}\dots z_n^{2g_n}]} \text{Coef}_{[x_1^0\dots x_n^0]} \prod_{i=1}^{n} \frac{1}{\mathcal{S}(z_i)} \prod_{k=1}^{s} \tilde{P}^{\text{loop}}(q) \prod_{k=s+1}^{r} \tilde{P}(\frac{x_{k_1}}{x_{k_2}}, q).$$

Again, we set

$$I_{\Gamma,\underline{g}}(q) = \sum_{\Omega} I_{\Gamma,\underline{g},\Omega}(q)$$

where the sum goes over all $n!$ orders of the vertices.

With this, we can generalize Theorem 2.2.13 to the version involving descendants:

Theorem 2.2.27 (Mirror symmetry for E, version with descendants)**.** *Fix $g \geq 2$, $n \geq 1$ and $k_1, \dots, k_n \geq 1$ satisfying $k_1 + \dots + k_n = 2g - 2$.*

We can express the series of descendant Gromov–Witten invariants of E in terms of Feynman integrals,

$$\sum_{d \geq 1} \langle \tau_{k_1}(pt) \dots \tau_{k_n}(pt) \rangle_{g,n}^{\mathcal{E},d} q^d = \sum_{(\text{ft}(\Gamma),\underline{g})} \frac{1}{|\operatorname{Aut}(\text{ft}(\Gamma), \underline{g})|} I_{\Gamma,\underline{g}}(q),$$

where Γ is a Feynman graph with a genus function $\underline{g}$, such that the vertex x_i has genus g_i and valency $k_i + 2 - 2g_i$, and such that $h^1(\Gamma) + \sum g_i = g$, and where we consider automorphisms of unlabeled graphs (ft is the forgetful map that forgets all labels of a Feynman graph Γ) respecting the genus function.

For a proof, see [18, 20].

2.3 Computation of Generating Series

In this section, we discuss our improved algorithm for computing tropical Hurwitz numbers and Gromov–Witten invariants via evaluation of Feynman integrals. The previous algorithm

is implemented in SINGULAR in the library `ellipticcovers.lib` [39]. It is based on evaluating the propagator product as a Laurent series in the variables x_i in the order given by Ω. For $k > s$, denoting the vertices adjacent to the (non-loop) edges q_k by x_{k_1} and x_{k_2} in any order, and fixed degree partition $\underline{a}$ over the base point, we evaluate

$$I_{\Gamma,\Omega,\underline{a}} = \mathrm{Coef}_{[x^0_{\Omega(1)}\cdots x^0_{\Omega(n)}]} \mathrm{Coef}_{\underline{a}} \prod_{k=1}^{s} P^{\mathrm{loop}}(q_k) \prod_{k=s+1}^{r} P\left(\frac{x_{k_1}}{x_{k_2}}, q_k\right)$$

by extracting the constant coefficient in the x_i and sum over all Ω. Vertex contributions are introduced as in Definition 2.2.26. For more details, see also [17].

2.3.1 Basic Improved Algorithm

Before turning to the flip signature, we first describe a structural improvement to the above mentioned method. Let us consider an edge labeled q_k in the graph Γ, connecting vertices x_i and x_j. The propagator associated with this edge will be a polynomial in the variable q_k. However, we are specifically interested in extracting the coefficient of the term $q_k^{2a_k}$, where a_k is the k-th entry of $\underline{a}$. Write $N = \sum_{i=1}^{3g-3} a_i$. By Lemma 2.2.18 and Equation (2.2), the coefficient of $q^{2\cdot\underline{a}}$ in $I_{\Gamma,\Omega}(q_1, \ldots, q_{3g-3})$ can explicitly be described as follows. For $a_k > 0$, we have

$$\sum_{w_k | a_k} w_k \left(\left(\frac{x_{k_1}}{x_{k_2}}\right)^{w_k} + \left(\frac{x_{k_2}}{x_{k_1}}\right)^{w_k} \right) = \frac{\sum\limits_{w_k | a_k} w_k (x_{k_1}^{N+w_k} x_{k_2}^{N-w_k} + x_{k_1}^{N-w_k} x_{k_2}^{N+w_k})}{x_{k_1}^N x_{k_2}^N}$$

and for $a_k = 0$, we obtain

$$\sum_{w_k=1}^{N} w_k \left(\frac{x_{k_1}}{x_{k_2}}\right)^{w_k} = \frac{\sum\limits_{w_k=1}^{N} w_k \cdot x_{k_1}^{N+w_k} x_{k_2}^{N-w_k}}{x_{k_1}^N x_{k_2}^N}.$$

In our computations, the focus lies on computing the numerator of this representation of the propagator. The denominator shift $x_i^N x_j^N$ can be taken into account separately within the function `coefficient_of_term` which determines the coefficient of a specific monomial $q^{2\cdot\underline{a}}$ by extracting the coefficient of the $x_i^N x_j^N$ term of the numerator.

2.3.2 Generating Series for Hurwitz Numbers

2.3.2.1 Flip Signature

In our approach, we introduce a flip signature with respect to the vertex ordering Ω to group covers having the same Feynman integral $I_{\Gamma,\Omega}$ together into a single computation. This grouping provides a further significant speedup of the computation of generating series of Gromov-Witten invariants via Feynman integrals $I_\Gamma = \sum_\Omega I_{\Gamma,\Omega}$ by identifying how many permutations Ω lead to the same signature, and determining the integral for one representative for each signature.

The permutation Ω determines which vertices of the edges will occur in the numerator and which will occur in the denominator of the corresponding propagator. While there are $(2g-2)!$ orders of branch points, there are at most 2^{2g-2} possible flip signatures, two possibilities for each edge of the graph. Moreover, the number of flip signatures is typically even smaller due to symmetries in the propagator product.

We define the flip signature as follows: We first find the *sign* of an edge q_k for a given vertex ordering Ω. We set the sign of an edge q_k to be 1, if $\Omega(i) < \Omega(j)$, otherwise it is set to be -1.

For a given graph Γ, order Ω, and branch type $\underline{a} = (a_1, \ldots, a_{3g-3})$, algorithm 1 `flip_signature` determines the flip signature as a vector of length $3g-3$ assigning each non-loop edge the value -1 in case the degree of the edge is zero and its sign is -1, and the value 0 otherwise. To detect loop edges we assign in this case the value -2.

2.3.2.2 Multiplicities

Algorithm 2 `signature_and_multiplicities` determines how often each flip signature occurs over all Ω. This is then called the *multiplicity* of the flip signature. Note that we only permute the vertices of edges, where a flip can occur. For the other vertices, the order will not change the flip signature. Hence, we count over a subgroup of the permutation group of all vertices, and therefore need to adjust the multiplicities of the flip signatures by the corresponding index.

Algorithm 1: `flip_signature`

Input: Graph Γ, permutation of vertices Ω, branch type $\underline{a}$.
Output: Flip signature b.

begin
  E = edges of Γ $(x_{k_1} - e_k - x_{k_2})$
  foreach e_k *in* E **do**
    foreach a_k *in* a **do**
      if $a_k = 0$ **then**
        if $\text{preimg}(\Omega, x_{k_1}) < \text{preimg}(\Omega, x_{k_2})$; `// preimg(Ω, x_k1) is the index of x_k1 in Ω`
        then
          $b_k = -1$ **else**
            $b_k = 0$
      if $x_{k_1} = x_{k_2}$ **then**
        $b_k = -2$
  return b

Algorithm 2: `signature_and_multiplicities`

Input: Graph Γ, branch type $\underline{a}$.
Output: Vector of flip signatures and their multiplicity

begin
  $E =$ edges of Γ $(x_{k_1} - e_k - x_{k_2})$
  $V =$ set of all vertices of edges k of Γ with $a_k = 0$
  $P = S(V)$ all permutations of V
  $D =$ empty dictionary
  foreach $\Omega \in P$ **do**
    $y = \text{sgn}(\Gamma, \Omega, a)$
    if y *key of* D **then**
      $D[y] = D[y] + 1$
    else
      $D[y] = 1$
  multiply each value of D by $\frac{(\#\text{ vertices of } \Gamma)!}{|P|}$
  return D

2.3.2.3 Feynman Integral

Building on signatures and multiplicities, we now proceed to the computation of the Feynman integral for a fixed branch type $\underline{a}$, which is one coefficient in the multivariate generating series and is denoted by $I_{\Gamma,\underline{a}}$. We also give an algorithm to compute the multivariate generating series up to a given degree d, which is denoted by $I_{\Gamma,d}$. The respective algorithms are Algorithm 3 `feynman_integral_branchtype` and Algorithm 4 `feynman_integral_degree`.

The functions `constterm` and `nonconstterm` compute the q-constant term of the propagator multiplied by $x_i^n x_j^n$ and the coefficient of q_k^{2d} of the propagator multiplied by $x_i^n x_j^n$, respectively. For given n they are given as

$$\text{constterm}(x_i, x_j, n) = \sum_{w=1}^{n} w \cdot x_i^{n+w} x_j^{n-w}$$

and

$$\text{nonconstterm}(x_i, x_j, q_k, d, n) = \sum_{w|d} w \cdot (x_i^{n+w} x_j^{n-w} + x_i^{n-w} x_j^{n+w}) \cdot q_k^{2d}.$$

Here we can choose $n = \sum_{k=1}^{3g-3} a_k$, since any variable x_i^d with degree $d < -n$ or $n < d$ can not contribute to the constant term $x_1^0 \cdot \ldots \cdot x_{2g-2}^0$.

Algorithm 3: `feynman_integral_branchtype`

Input: Graph Γ, branch type $\underline{a}$.
Output: $I_{\Gamma,\underline{a}}$

begin
  $E =$ edges of Γ
  $N = \sum_{i=1}^{3g-3} a_i$
  $D = \text{signatures\_and\_multiplicities}(\Gamma, \underline{a})$
  $p = 0$
  foreach $f \in \text{keys}(D)$ **do**
    $\text{tmp} = 1$
    for j **from** 1 **to** $\text{size}(f)$ **do**
      if $k_j = -1$ **then**
        $\text{tmp} = \text{tmp} \cdot \text{constterm}(x_{E_{j,1}}, x_{E_{j,2}}, N)$
      else if $k_j = 0$ **then**
        $\text{tmp} = \text{tmp} \cdot \text{constterm}(x_{E_{j,2}}, x_{E_{j,1}}, N)$
      else
        $\text{tmp} = \text{tmp} \cdot \text{nonconstterm}(x_{E_{j,1}}, x_{E_{j,2}}, q_j, f_j, N)$
    $p = p + D(f) \cdot \text{Coef}_{[x_1^N \cdots x_{2g-2}^N]}(\text{tmp})$
  return p

Algorithm 4: `feynman_integral_degree`

Input: Graph Γ, maximum degree d.
Output: $I_{\Gamma,d}$

```
1 begin
2     sum = 0;
3     for j from 1 to d do
4         A = partitions of d into #(edges of Γ) summands.
5         foreach a ∈ A do
6             sum += feynman_integral_branchtype(Γ, a)
7     return sum
```

2.3.3 Generating Series for Descendant Gromov-Witten Invariants

The improved algorithms from Section 2.3.1 have a straight-forward generalization to the case of Gromov–Witten invariants with psi-classes, following the formula given at the beginning of Section 2.3. We omit the respective algorithms here, but give example and timings in [40].

2.4 Timings

In this section, we evaluate the performance of our implementation for various graphs and degrees d in the Tables 2.1 and 2.2. We compare

- the implementation of the original algorithm in the SINGULAR library `ellipticcovers.lib` in the function `gromovWitten` (denoted by Singular-1)
- the implementation of the new algorithm in the function `feynman_integral` included in an updated version of `ellipticcovers.lib` (denoted by Singular-2); and
- the respective OSCAR-based implementation in the function `feynman_integral` of the package `GromovWitten.jl` [40] (denoted by OSCAR).

The package `GromovWitten.jl` is available at

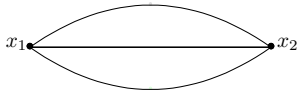
`https://github.com/singular-gpispace/GromovWitten`.

All computations have been done sequentially on an M-core with 3.2 MHz. We have verified that all answers are consistent. The tables show all data collected in the respective configuration. All numbers are in seconds rounded to 3 digits.

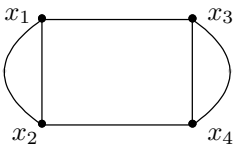
We consider the source curves depicted in Figures 2.2a–2.3b. To generate the graphs considered in the timings, the following code can be used:

```
julia> using GromovWitten

# caterpillar genus 2
julia> G =  feynman_graph([(1,2),(1,2),(1,2)]);

# caterpillar genus 3
julia> G = feynman_graph([(1,2),(1,2),(2,4),(1,3),(3,4),(3,4)]);

# caterpillar genus 4
julia> G = feynman_graph([(1,2),(1,2),(1,3),(2,4),(3,4),(3,5),(4,6),(
↪  5,6),(5,6)]);

# star graph K_{1,3}
julia> G = feynman_graph([(1,2),(1,3),(1,4),(2,4),(2,3),(3,4)]);
```

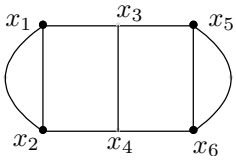
For the timings, see Tables 2.1 and 2.2, and for a visualization of the timings, see Figures 2.4 and 2.5. We observe that the new algorithm presented in this paper significantly outperforms the previous one, and that the OSCAR implementation of that algorithm significantly outperforms the SINGULAR implementation again, although it is structurally identical. The demonstrates the potential of the new OSCAR platform. The OSCAR implementation allows one to compute, in comparable time, at least double the degree in the expansion of the formal generating series than the SINGULAR implementation, which in turn can achieve about three times the degree compared to the old algorithm. The at least six-fold or more increase in degree allows for example to study homogeneity properties of quasi modular representations of the generating series for many graphs of larger genus which were previously not tractable. There is ongoing work on parallelization of the computation.



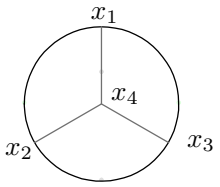
(A) Caterpillar genus 2.



(B) Caterpillar genus 3.



(A) Caterpillar genus 4.



(B) Star graph $K_{1,3}$.

Degree d	Caterpillar genus 2			Caterpillar genus 3		
	Singular-1	Singular-2	OSCAR	Singular-1	Singular-2	OSCAR
1	0.01	0.01	0.00001	0.10	0.10	0.0003
2	0.70	0.02	0.00006	0.74	0.35	0.001
3	1.23	0.021	0.0001	10.20	0.78	0.004
4	10.21	0.030	0.00016	72.41	1.54	0.012
5	51.84	0.032	0.0002	343.03	2.76	0.03
6	173.94	0.041	0.0003	1 300	4.72	0.68
7	345.16	0.044	0.00035	4 030	7.94	0.14
8	570.06	0.051	0.0004	—	13.6	0.26
9	819.31	0.053	0.0005	—	23.9	0.47
10	1 060	0.06	0.0007	—	41.9	0.82
11	1 287	0.07	0.0007	—	72.3	1.35
12	1 386	0.08	0.0010	—	122	2.24
13	1 714	0.09	0.0011	—	211	3.48
14	2 433	0.10	0.0014	—	351	5.62
15	3 376	0.11	0.0015	—	559	8.49
16	4 390	0.12	0.0018	—	940	12.9
17	—	0.14	0.0019	—	1 450	20.7
18	—	0.15	0.0023	—	2 300	27.6
19	—	0.17	0.0024	—	3 440	40.5
20	—	0.19	0.0029	—	5 130	58.7

TABLE 2.1: Timings for caterpillar source curves of genus 2 and genus 3.

2.4.1 Example Computation

In this section, we provide example code for computing the Hurwitz numbers for a caterpillar genus 3 source curve.

```
julia> using GromovWitten

julia> G = feynman_graph([(1, 3),(1, 2),(1, 2),(2, 4),(3, 4),(3, 4)])
FeynmanGraph([(1, 3), (1, 2), (1, 2), (2, 4), (3, 4), (3, 4)])

julia> F = feynman_integral(G);

julia> a = [0, 2, 1, 0, 0, 1];
```

degree d	Caterpillar genus 4			Star graph $K_{1,3}$		
	Singular-1	Singular-2	OSCAR	Singular-1	Singular-2	OSCAR
1	32.16	8.24	0.021	0.08	0.17	0.0007
2	4016	35.0	0.12	1.20	0.42	0.003
3	–	136	0.83	18.1	0.92	0.012
4	–	2240	5.53	124	1.77	0.039
5	–	32 890	29.0	549	3.19	0.112
6	–	–	121	1 990	5.80	0.27
7	–	–	419	6 080	10.3	0.56
8	–	–	2 420	18 200	19.7	1.10
9	–	–	3 490	–	38.0	1.94
10	–	–	38 700	–	72.0	3.21
11	–	–	–	–	130	5.28
12	–	–	–	–	238	8.50
13	–	–	–	–	424	13.8
14	–	–	–	–	716	20.2
15	–	–	–	–	1 190	28.6
16	–	–	–	–	1 880	40.8
17	–	–	–	–	2 970	57.3
18	–	–	–	–	4 777	79.2
19	–	–	–	–	7 103	107
20	–	–	–	–	10 620	144

TABLE 2.2: Timings for caterpillar genus 4 and $K_{1,3}$ source curves.

```
julia> feynman_integral_branch_type(F, a)
256*q[2]^4*q[3]^2*q[6]^2

julia> feynman_integral_degree(F, 3)
288*q[1]^6 + 32*q[1]^4*q[2]^2 + 32*q[1]^4*q[3]^2 + 32*q[1]^4*q[5]^2 +
↪  32*q[1]^4*q[6]^2 + 8*q[1]^2*q[2]^2*q[5]^2 + 8*q[1]^2*q[2]^2*q[6]^2
↪  + 8*q[1]^2*q[3]^2*q[5]^2 + 8*q[1]^2*q[3]^2*q[6]^2 + 24*q[2]^6 +
↪  152*q[2]^4*q[3]^2 + 8*q[2]^4*q[5]^2 + 8*q[2]^4*q[6]^2 +
↪  152*q[2]^2*q[3]^4 + 32*q[2]^2*q[3]^2*q[5]^2 +
↪  32*q[2]^2*q[3]^2*q[6]^2 + 32*q[2]^2*q[4]^4 +
↪  8*q[2]^2*q[4]^2*q[5]^2 + 8*q[2]^2*q[4]^2*q[6]^2 + 8*q[2]^2*q[5]^4
↪  + 32*q[2]^2*q[5]^2*q[6]^2 + 8*q[2]^2*q[6]^4 + 24*q[3]^6 +
↪  8*q[3]^4*q[5]^2 + 8*q[3]^4*q[6]^2 + 32*q[3]^2*q[4]^4 +
↪  8*q[3]^2*q[4]^2*q[5]^2 + 8*q[3]^2*q[4]^2*q[6]^2 + 8*q[3]^2*q[5]^4
↪  + 32*q[3]^2*q[5]^2*q[6]^2 + 8*q[3]^2*q[6]^4 + 288*q[4]^6 +
↪  32*q[4]^4*q[5]^2 + 32*q[4]^4*q[6]^2 + 24*q[5]^6 +
↪  152*q[5]^4*q[6]^2 + 152*q[5]^2*q[6]^4 + 24*q[6]^6

julia> f = feynman_integral_degree_sum(F, 3)
```

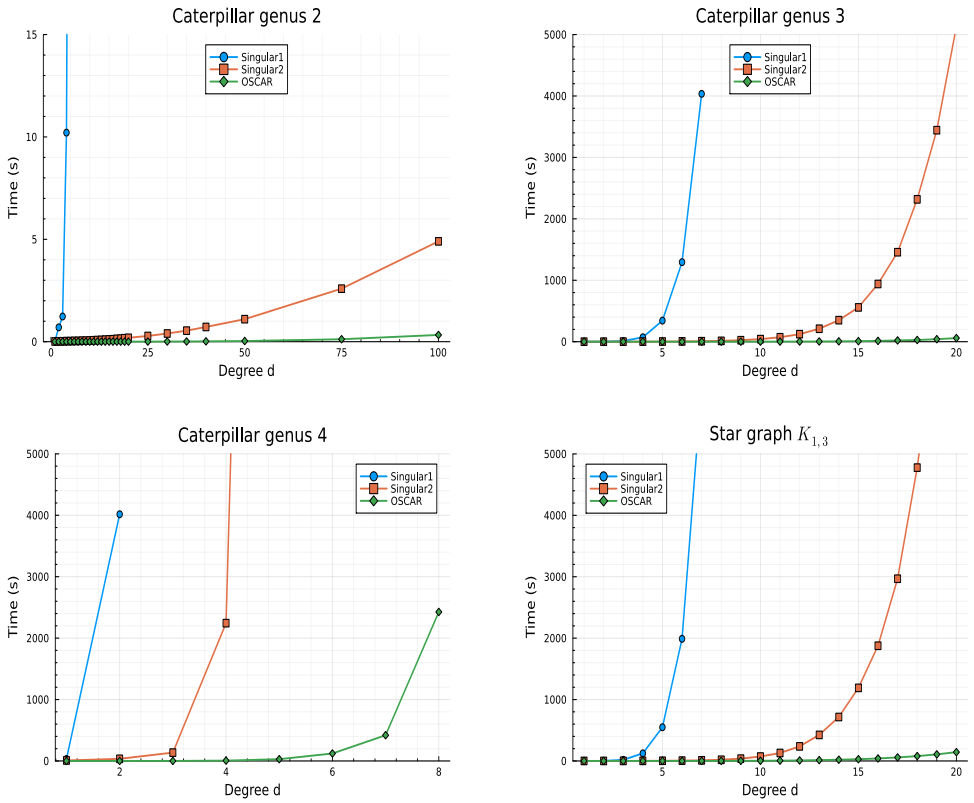


FIGURE 2.4: Visualization of the timings for Singular-1, Singular-2, and OSCAR.

```
288*q[1]^6 + 32*q[1]^4*q[2]^2 + 32*q[1]^4*q[3]^2 + 32*q[1]^4*q[5]^2 +
↪  32*q[1]^4*q[6]^2 + 8*q[1]^4 + 8*q[1]^2*q[2]^2*q[5]^2 +
↪  8*q[1]^2*q[2]^2*q[6]^2 + 8*q[1]^2*q[3]^2*q[5]^2 +
↪  8*q[1]^2*q[3]^2*q[6]^2 + 24*q[2]^6 + 152*q[2]^4*q[3]^2 +
↪  8*q[2]^4*q[5]^2 + 8*q[2]^4*q[6]^2 + 152*q[2]^2*q[3]^4 +
↪  32*q[2]^2*q[3]^2*q[5]^2 + 32*q[2]^2*q[3]^2*q[6]^2 +
↪  8*q[2]^2*q[3]^2 + 32*q[2]^2*q[4]^4 + 8*q[2]^2*q[4]^2*q[5]^2 +
↪  8*q[2]^2*q[4]^2*q[6]^2 + 8*q[2]^2*q[5]^4 + 32*q[2]^2*q[5]^2*q[6]^2
↪  + 8*q[2]^2*q[6]^4 + 24*q[3]^6 + 8*q[3]^4*q[5]^2 + 8*q[3]^4*q[6]^2
↪  + 32*q[3]^2*q[4]^4 + 8*q[3]^2*q[4]^2*q[5]^2 +
↪  8*q[3]^2*q[4]^2*q[6]^2 + 8*q[3]^2*q[5]^4 + 32*q[3]^2*q[5]^2*q[6]^2
↪  + 8*q[3]^2*q[6]^4 + 288*q[4]^6 + 32*q[4]^4*q[5]^2 +
↪  32*q[4]^4*q[6]^2 + 8*q[4]^4 + 24*q[5]^6 + 152*q[5]^4*q[6]^2 +
↪  152*q[5]^2*q[6]^4 + 8*q[5]^2*q[6]^2 + 24*q[6]^6

julia> substitute(f)
1792*q[1]^6 + 32*q[1]^4
```

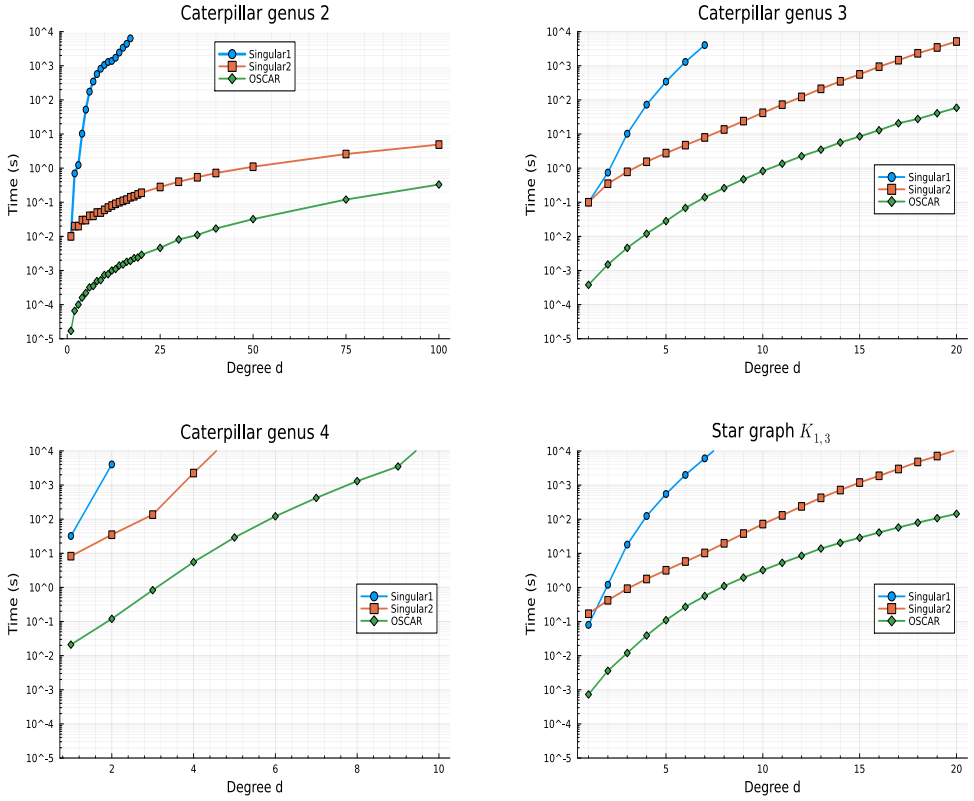


FIGURE 2.5: Visualization of the timings for Singular-1, Singular-2, and OSCAR (logarithmic time scale).

Chapter 3

Towards Parallel Algorithms for Gromov-Witten Invariants of Elliptic Curves

We present a parallel enhanced algorithm for exploring mirror symmetry for elliptic curves through the correspondence of algebraic and tropical geometry, focusing on Gromov-Witten invariants of elliptic curves and, in particular, Hurwitz numbers. We present a new highly efficient algorithm for computing generating series for these numbers. A sequential version of the algorithm has been implemented using Singular and OSCAR. The implementations outperform by far the previous methods provided in Singular. In this note, we describe work towards the natural next step, which is parallelization. We have integrated our algorithm with GPI-Space, a workflow management system for high-performance computing developed at Fraunhofer ITWM. This allows us to run our algorithm simultaneously on a large number of cores. This facilitates computation of quasi-modular representations of the respective generating series.

3.1 Introduction

This chapter centers on a use case of the principle of mirror symmetry, which establishes a connection between Gromov-Witten invariants of an elliptic curve and certain integrals over Feynman graphs. Our goal is to provide parallel algorithms for computing Gromov-Witten invariants of elliptic curves. Tropical methods fit into the picture of mirror symmetry through the famous Gross-Siebert program [32].

Here, we present the special case of elliptic curves, for which mirror symmetry is best understood [16]. The tropical side of mirror symmetry for elliptic curves was discovered within the Collaborative Research Center TRR195, which is also responsible for the development of OSCAR [17, 18, 24].

One consequence of mirror symmetry for elliptic curves is the equality of the generating function of Hurwitz numbers (resp. more generally, of descendant Gromov-Witten invariants) to certain Feynman integrals which are complex analytic path integrals whose construction is governed by combinatorics. In this chapter, we first recall our enhanced algorithm, which has been implemented in [28]. This algorithm computes generating series of Hurwitz numbers via Feynman integrals. Hurwitz numbers enumerate branched covers of non-singular curves with a specified ramification profile over fixed points. One key application is to find the representation of the generating series as quasi-modular forms. This requires solving a linear system of equations which becomes determined if we know Hurwitz numbers of large enough degree. In the case of descendant Gromov-Witten invariants, homogeneity of the quasi-modular form is an interesting question. To expedite the computation, which can be very time-consuming for covers of high degree and graphs of high genus, we utilize GPI-Space, a Petri net-based workflow management system developed by the HPC group at Fraunhofer ITWM, to parallelize our algorithm.

3.2 Gromov-Witten Invariants and Feynman Integrals

3.2.1 Theoretical Background and Elementary Algorithm

Let $\mathcal{E}$ be an arbitrary complex elliptic curve and $\mathcal{C}$ a non-singular curve of genus g and $\phi : \mathcal{C} \to \mathcal{E}$ a cover.

Definition 3.2.1 (Hurwitz numbers)**.** Fix $2g-2$ points $p_1, \dots, p_{2g-2}$ in $\mathcal{E}$. We define the Hurwitz number $N_{d,g}$ to be the weighted number of (isomorphism classes of) simply ramified covers $\phi : \mathcal{C} \to \mathcal{E}$ of degree d, where $\mathcal{C}$ is a connected curve of genus g, and the branch points of ϕ are the points p_i, $i = 1, \dots, 2g-2$. We count each such cover ϕ with weight $|\operatorname{Aut}(\phi)|$.

Lemma 3.2.2. *Fix a Feynman graph Γ and an order Ω , and a tuple $(q_1, \dots, q_{3g-3})$ as in Definition 2.2.12. We express the coefficient of $q^{2\cdot\underline{a}}$ in $I_{\Gamma,\Omega}(q_1, \dots, q_{3g-3})$. Assume k is such that the entry $a_k = 0$, and assume the edge q_k connects the two vertices x_{k_1} and x_{k_2}. Choose the notation of the two vertices x_{k_1} and x_{k_2} such that the chosen order Ω implies $\left|\frac{x_{k_1}}{x_{k_2}}\right| < 1$ for the starting*

points on the integration paths. Then the coefficient of $q^{2\cdot \underline{a}}$ equals the constant term of the series

$$\prod_{k|a_k=0}\left(\sum_{w_k=1}^{\infty} w_k\cdot\left(\frac{x_{k_1}}{x_{k_2}}\right)^{w_k}\right)\cdot\prod_{k|a_k\neq 0}\left(\sum_{w_k|a_k} w_k\cdot\left(\left(\frac{x_{k_1}}{x_{k_2}}\right)^{w_k}+\left(\frac{x_{k_2}}{x_{k_1}}\right)^{w_k}\right)\right) \tag{3.1}$$

For a series

$$F(x_1,\ldots,x_n)=\sum_{\substack{1\leq j\leq n\\ a_j\geq 0}}\alpha(a_1,\ldots,a_n)x_1^{a_1}\cdots x_n^{a_n} \tag{3.2}$$

we write

$$\operatorname{coeff}_{[x_1^{a_1},\ldots,x_n^{a_n}]}(F)=\alpha(a_1,\ldots,a_n). \tag{3.3}$$

Definition 3.2.3 (The propagator and Feynman integral)**.** We define the propagator as a (formal) series in x and q,

$$P(x,q)=\sum_{w=1}^{\infty} w\cdot x^w+\sum_{a=1}^{\infty}\left(\sum_{w|a} w(x^w+x^{-w})\right)q^a \tag{3.4}$$

Fix a Feynman graph Γ and an order Ω of vertices and write

$$P_{\Gamma,\Omega}=\prod_{k=1}^{3g-3}\left(-P\left(\frac{x_{k_1}}{x_{k_2}},q\right)\right).$$

Then we define the Feynman integral as

$$I_{\Gamma,\Omega}(q)=\operatorname{coeff}_{[x_1^0,\ldots,x_{2g-2}^0]}(P_{\Gamma,\Omega}).$$

3.2.2 Improvement of the algorithm

In the standard algorithm, the Feynman integral is computed for a given branch type a, by computing the propagator of a for each permutation of vertices, while the enhanced algorithm works by assigning to each cover a so-called flip signature by considering the permutation assigning vertices of the source tropical curve to branch points on the target elliptic curve, observing that covers with the same signature lead to the same integral. Further improvements are achieved by direct computation of coefficients of the Laurent series of each propagator, avoiding explicit expansion of all propagators together and then compute the coefficient. The enhanced algorithm outperforms by far the standard algorithm and has been implemented in OSCAR.

Consider the example of a source curve in Figure 3.1.

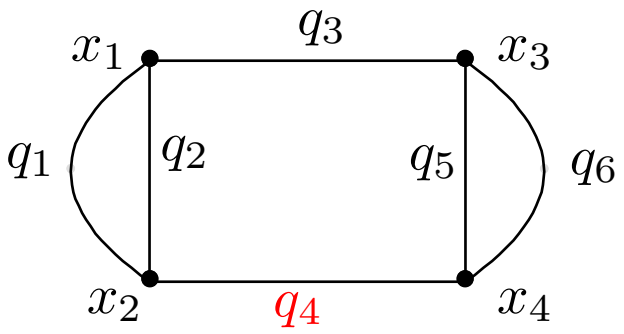


FIGURE 3.1: Caterpillar genus 3

$a = (1, 1, 2, 0, 3, 1), \Omega = (1, 3, 4, 2)$ here $x_4 < x_2$, $d = \text{sum}(a)$.

Construction 3.2.1 (Standard Algorithm)**.** In the standard algorithm, the propagator is calculated for every possible permutation of the order of vertices.

```
P = permutations(O) # 4! elements.
for p in P
    Propagator *= propagator(G, a, p)
    # For each p, we check if $x_4 < x_2$.
end
```

The Feynman integral at branch a is then calculated as:

$$I_q(\Gamma, a) = \text{coeff}_{[x_1^0,\ldots,x_4^0]} \text{propagator}$$

Construction 3.2.2 (Enhanced Algorithm)**.** In the enhanced algorithm, we first compute the flip_signature and group all vertex orders that lead to the same branch type a.

$$\text{flip\_signature}(a, p) = \begin{cases} 12 \times (1, 1, 2, 0, 3, 1) & \text{if } x_4 < x_2 \\ 12 \times (1, 1, 2, -1, 3, 1) & \text{if } x_4 > x_2 \end{cases}$$

For example, vertex orders lead to either $(1, 1, 2, 0, 3, 1)$ or $(1, 1, 2, -1, 3, 1)$. We then compute the local Feynman integral:

$$I_{q,1}(\Gamma, a) = \text{coeff}_{[x_1^0],\ldots,x_4^0} \text{propagator}(\Gamma, (1, 1, 2, 0, 3, 1))$$

$$I_{q,2}(\Gamma, a) = \text{coeff}_{[x_1^0],\ldots,x_4^0} \text{propagator}(\Gamma, (1, 1, 2, -1, 3, 1))$$

Finally, the Feynman integral at branch a is obtained by summing $I_{q,1}$ and $I_{q,2}$:

$$I_q(\Gamma, a) = I_{q,1} + I_{q,2}$$

The computation becomes complicated and slow when we attempt to calculate the degree of a Feynman integral for a given graph Γ with a fixed degree d.

To address this issue, we utilize GPI-Space, a High-Performance Computing (HPC) system currently under development at ITWM Fraunhofer Kaiserslautern. This enables us to parallelize our algorithm, allowing us to compute the Feynman integral simultaneously for all combinations a of degree d ($d = \sum(a)$).

3.3 Petri nets and GPI-Space

While sequential algorithms are typically described in a step-by-step manner, Petri nets offer a representation that mirrors both the structure of the algorithms and the current state of computation. Consequently, they provide a means to automatically exploit parallel structures. A Petri net is essentially a directed bipartite graph with two types of vertices: places (depicted as circles) and transitions (illustrated as rectangular boxes). The transitions represent elementary functional units, while the places can contain marked tokens, which are akin to representing data pieces (in the context of colored Petri nets). These transitions serve to connect places, consuming one token from each input place and depositing one token onto each output place. As such, a transition is only capable of firing if all input places hold a token. Figure 3.2 presents an example of a Petri net.

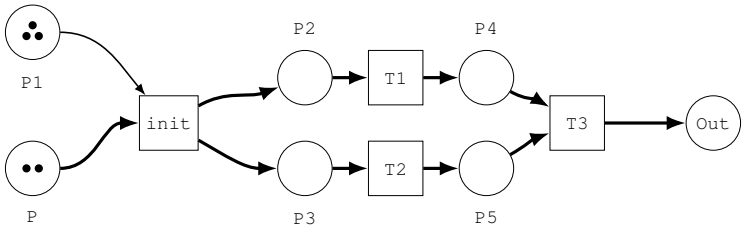


Figure 3.2: Task and data parallelism in a Petri net

3.4 Parallel enumeration of combinatorial objects

To compute the Feynman integral of degree d, the typical approach involves partitioning d into n edge parts, resulting in

$$\binom{d+n-1}{d}$$

elements. However, this partitioning process and the subsequent computation can be time-consuming.

To address this efficiently, we first compute d elements $(b_1, \ldots, b_d)$ of the partition using the gen_block(n, d) function (see Algorithm 5). For each element b_i, we compute its next partition, denoted as m_i, using the iterate(b_i) function (see Algorithm 7). This parallel computation strategy significantly improves the effectiveness of computing the Feynman integral.

Algorithm 5: gen_block

Input: Integer n, Integer d.
Output: Vector of vectors ru.

```
1 begin
2     ru = empty vector of vectors;
3     for e = 0 to d − 1 do
4         x = [d − e; fill(0, n − 2); e];
5         push x to ru;
6     return ru;
```

Algorithm 6: next_partition

Input: Vector a representing a partition.
Output: Next partition.

```
1  begin
2      n = sum(a);
3      k = length(a);
4      for i = k to 1 do
5          if i = k and a[i] = n then
6              return a;
7          else
8              for j = i − 1 to 1 do
9                  if a[j] ≠ 0 then
10                     a[j]− = 1;
11                     a[k] = 0;
12                     a[j + 1] = ak + 1;
13                     return a;
```

Algorithm 7: iterate

Input: vector x.

Output: Vector of vectors ru.

```
begin
    k = length(x);
    d = sum(x);
    e = d − x[1];
    n = binomial(d+k−1, d);
    y = [x[1] − 1; 0; ...; 0; e + 1];
    for i = 1 to n do
        if x ≠ y then
            x = next_partition(x);
            push x to ru;
        else
            Break;
    vec = [d; 0; ...; 0];
    if x == vec then
        push x to the front of ru;
    return ru;
```

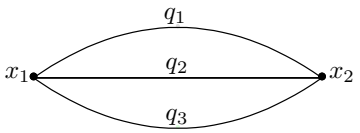


FIGURE 3.3: Caterpillar genus 2

Example 3.4.1. *Suppose $d = 4$ and $n = 3$ and we consider as a small example the graph in Figure 3.3. Table 3.4 illustrates how we can parallelize our algorithm.*

Utilizing GPI-SPACE alongside the Petri net depicted in Figure 3.5 enables us to compute the `feynman_integral(G,a)` by independently computing each row in the table in Figure 3.4 determining a from the preceding one. Subsequently, we aggregate these computations at `feynman_degree_sum(G,d)` which, in the example, gives 720.

gen_block(d,n)	iterate(vec)	feynman_integral(G,a)
[4, 0, 0]	[4, 0, 0]	84
	[3, 1, 0]	40
[3, 0, 1]	[3, 0, 1]	40
	[2, 2, 0]	68
	[2, 1, 1]	8
[2, 0, 2]	[2, 0, 2]	68
	[1, 3, 0]	40
	[1, 2, 1]	8
	[1, 1, 2]	8
[1, 0, 3]	[1, 0, 3]	40
	[0, 4, 0]	84
	[0, 3, 1]	40
	[0, 2, 2]	68
	[0, 1, 3]	40
	[0, 0, 4]	84

FIGURE 3.4: Parallel enumeration of combinations leading to parallelization of Feynman integral evaluation

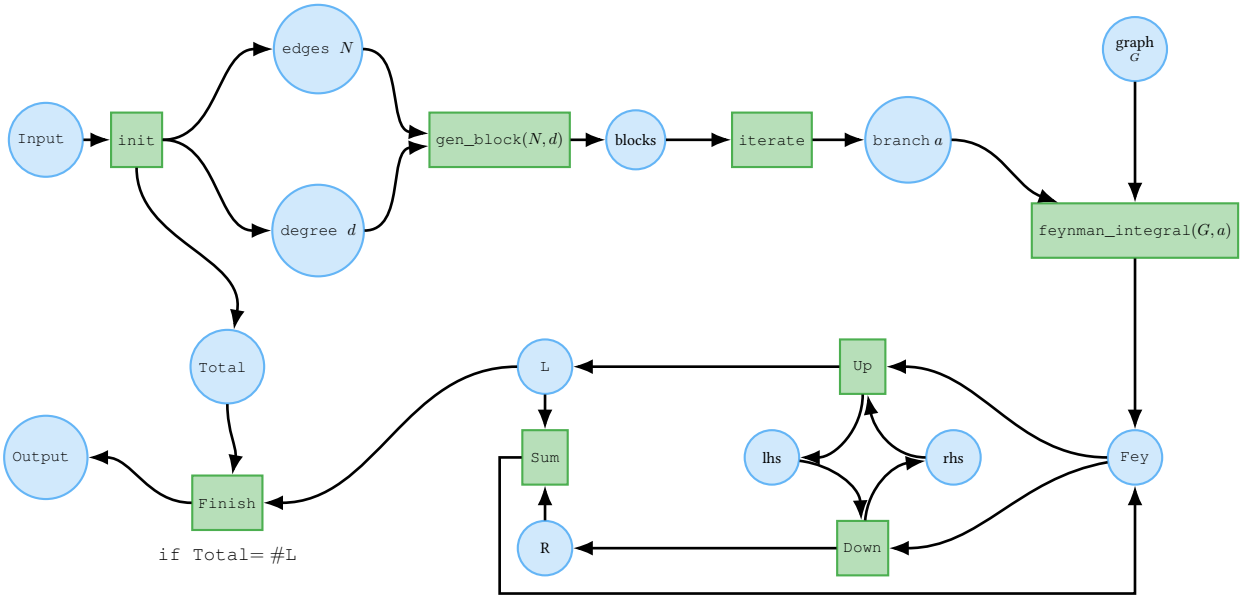


FIGURE 3.5: Petri-net

Chapter 4

Parallel Computation of Gromov-Witten Invariants

4.1 Introduction

In this chapter, we present a new parallelized version of the Feynman Integral over the vectors of the branch type $\mathbf{a} = (a_1, \ldots, a_n)$. Our approach involves incorporating a fixed amount m of vectors $\mathbf{a}$ at a specific place within a Petri-net structure. By utilizing the parallel processing capabilities of GPI-Space, we demonstrate how this method can significantly outperform traditional `for` loop iterations over simple numerical vectors or integers. Furthermore, we extend our analysis by comparing the performance of our parallelized approach in a more general setting, where parallelization is applied across all branch type vectors $\mathbf{a}$. This comparative study will highlight the efficiency gains achievable through parallel computation in the context of Feynman Integrals.

> *All computations in this work (including the current and next chapters) were performed on a computer equipped with an Intel® Core™ i7-9700 processor (8 cores @ 3.00 GHz, 16 GB DDR4 RAM) running Ubuntu 22.04.5 LTS.*

4.2 Parallel Computing with GPI-Space

4.2.1 How Cores Affect Speed

The speed of GPI-Space strongly depends on the number of workers (cores) available on the computer. The more cores there are, the faster the code runs. In Figure 4.1, we provide a plot that illustrates how the computation time varies with the number of cores.

Example 4.2.1. *Here, for a fixed graph, we compare the timing of our algorithm implemented in C++ and GPI-Space as a function of the number of cores.*

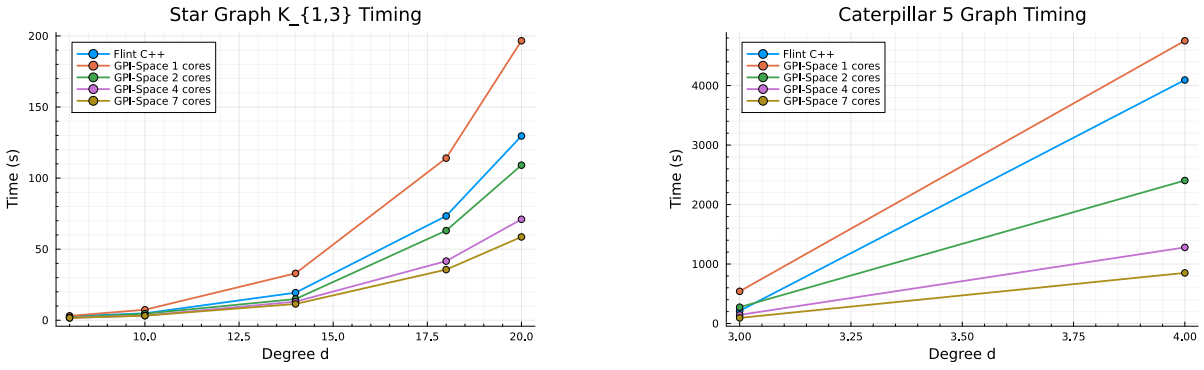


Figure 4.1: Computation time as a function of the number of cores for both the C++ implementation and GPI-Space.

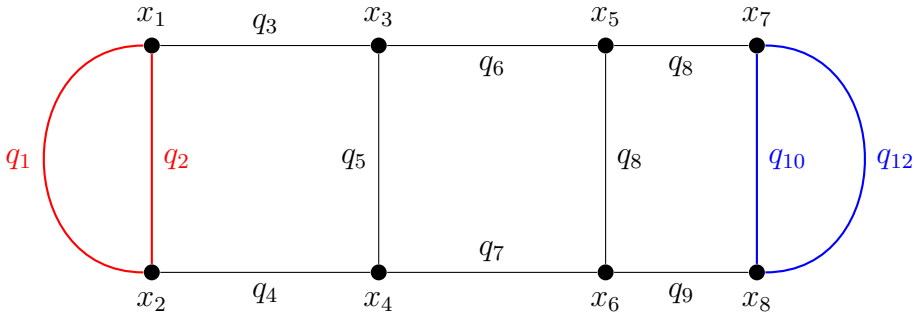


Figure 4.2: Caterpillar genus 5

4.2.2 Comparison of C++ and Julia Code

We rewrote the original code, written in Julia, in C++. The new implementation heavily relies on the FLINT (Fast Library for Number Theory) library, which significantly boosts the code's performance. By transitioning to C++, we also took advantage of GPI-Space for parallel computing, resulting in significant performance and efficiency improvements.

In Table 4.1, 4.2 and 4.3, we present a comparison of execution times for various graphs. The results clearly show that the C++ implementation is faster than the original Julia code.

Table 4.1: Caterpillar Graph of Genus 4

Graph	Degree	Flint-C++	Julia
Caterpillar	4	12.03	14.75
	5	56.65	85.18
	6	223.63	377.37
	7	813.28	1343.18
	8	2342.51	4024.92

Table 4.2: Star Graph $K_{1,3}$

Graph	Degree	Flint-C++	Julia
Star Graph $K_{1,3}$	8	2.23	1.53
	14	19.38	28.93
	20	129.63	225.54
	22	222.65	389.95
	25	457.25	829.64

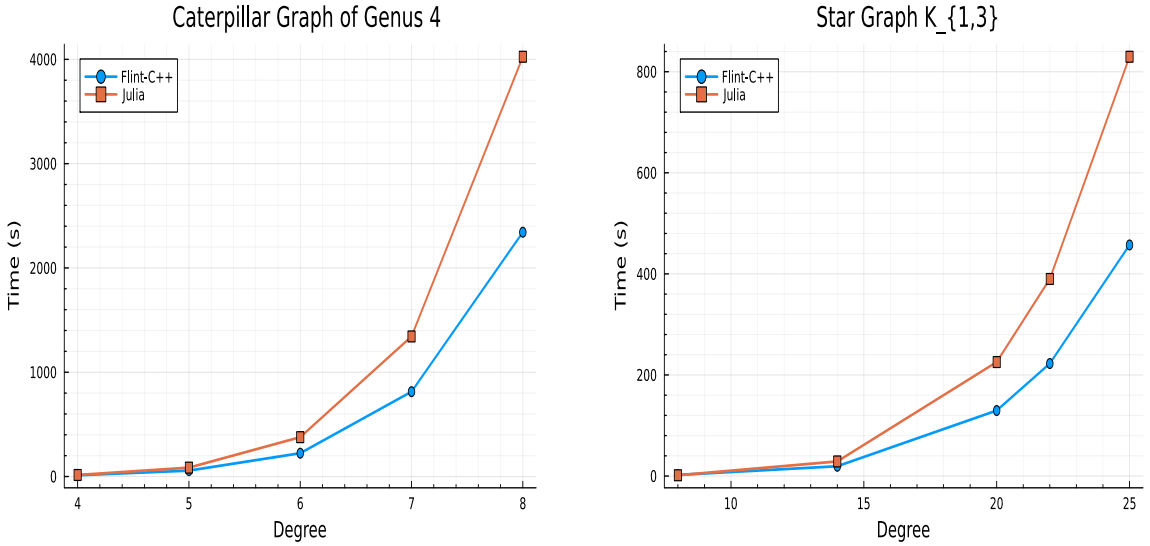


Figure 4.3: Visualization of the timings for Julia and Flint-C++

4.2.3 Algorithms

In Chapter 2, we presented a code implementation for computing Hurwitz numbers using Feynman integrals, initially developed in Julia and integrated within the Oscar software system. To enable parallelization of this computation using GPI-Space, we have rewritten the code in C++ using the FLINT library [41]. Following the methodology outlined in [29], we redesigned the partition algorithm, referred to as `partition(n, d)`, where n represents the number of edges and d is the degree $d = \sum a_i$. This modification facilitates the parallel execution of the algorithm.

Our parallelization strategy begins with developing an algorithm that generates the next partition of a given vector. For the initial implementation in GPI-Space, we compute the first vector $\mathbf{a}$ of length n and sum $d = \sum a_i$, initialized as $\mathbf{a} = [d, 0, \ldots, 0]$. We then allocate a fixed number m of `next_partition`($\mathbf{a}$) computations to a designated place within the Petri net.

In the next transition, we compute the Feynman integral for these m vectors and then proceed to generate the next m vectors for further Feynman integral calculations. This approach allows us to compute a fixed number of m vectors in parallel, as opposed to a classical for loop that processes each vector sequentially.

The second parallelization strategy focuses on dividing the initial vector $\mathbf{a}^0 = [d, 0, \ldots, 0]$ into $d + 1$ distinct vectors, specifically $\mathbf{a}^1 = [d - 1, 0, \ldots, 0, 1]$, $\mathbf{a}^i = [d - i, 0, \ldots, 0, i]$, up to $\mathbf{a}^d = [0, \ldots, 0, d]$, using the `gen_block` (algorithm 5).

For each vector $\mathbf{a}^i$ generated by `gen_block`, the algorithm `iterate`($\mathbf{a}^i$) provides $\binom{N+i}{i}$ partitions, where $N = n - 2$.

In the parallelized implementation, each partition derived from `iterate`($\mathbf{a}^i$) is used to compute the Feynman integral. Specifically, the algorithm generates $\binom{N+i}{i}$ vectors from `iterate`($\mathbf{a}^i$). For each of these vectors, the Feynman integral is calculated. These computations are performed in parallel, utilizing the capability of GPI-Space to handle concurrent processes efficiently.

Thus, for every vector $\mathbf{a}^i$ obtained from `gen_block`, a parallel computational task is established where the Feynman integral for each vector is computed simultaneously. This parallel approach ensures that the total computation time is significantly reduced by the use of multiple processors to handle a huge number of Feynman integrals.

In Chapter 3, the `partitions`(n,d) function is constructed by combining the `gen_block`(n,d) algorithm (see Algorithm 5) with the `iterate`(a) function (Algorithm 7), which in turn relies on `next_partition` (Algorithm 6).

Here, we rewrite the Feynman integral of a given degree, defined in Algorithm 4, using the `gen_block` and `iterate` algorithms.

Algorithm 8: feynman_integral_degree

Input: Graph G, degree d
Output: Sum of Feynman integrals over all branch types

```
1 begin
2     Initialize sum = 0;
3     n ← # Edges(G);
4     gen ← gen_block(n, d);
5     foreach x in gen do
6         it ← iterate(x);
7         foreach x_i in it do
8             sum += feynman_integral_branch_type(G, x_i);
9     return sum;
```

4.2.3.1 Necessity of parallelism

Parallelism becomes essential due to the size $\binom{n+d-1}{d}$ of the partition set, denoted as partition(n, d).

As the values of n (which represents the number of edges) and d (which represents the degree) increase, the size of this partition set grows significantly. This leads to substantial computational complexity for the functions `gen_block(n, d)` and `iterate`.

4.3 Petri nets

4.3.1 Petri nets

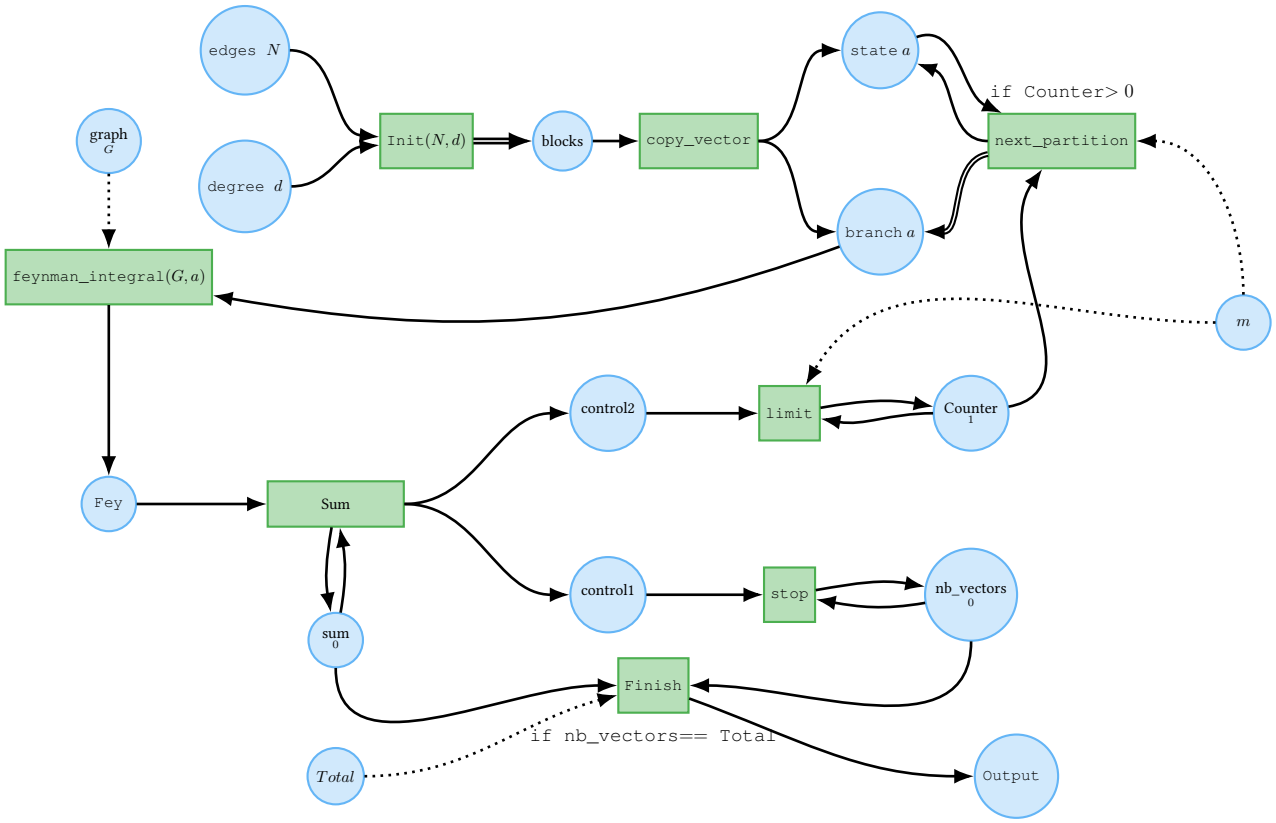


FIGURE 4.4: Petri-net

Explanation of the Petri Net

The Petri net described here is designed to compute Feynman integrals in parallel by processing a fixed number of m branch-type vectors.

Initialization (`Init` Transition)

The process begins with the `Init` transition, which generates the initial vector $\mathbf{a} = [d, 0, \ldots, 0]$ of length N. This vector represents the starting point for the computations.

Copying Vectors (`copy_vector` Transition)

The initial vector $\mathbf{a}$ is then copied into two places: `branch` and `state`.

- `branch` holds the current vector that will be processed.
- `state` retains the most recent vector, which will be used to generate the next set of vectors.

Generating Next Partitions (`next_partition` Transition)

The `next_partition` transition computes the next m vectors using the `next_partition` algorithm and places them into the `branch` place. The mth vector generated in this process is stored in `state` to be used as the starting point for the next round of vector generation.

Parallel Computation of Feynman Integrals (`feynman_integral` Transition)

All vectors in the `branch` place are used in parallel to compute the Feynman integral feynman_integral$(G, \mathbf{a})$. This parallel processing is a critical component, allowing for the simultaneous calculation of multiple integrals.

Summing and Counting (`Sum` Transition)

The `Sum` transition counts the number of Feynman integrals computed. A feedback loop is established using the `counter` place, which informs the `next_partition` transition when m integrals have been computed. This feedback ensures that the `next_partition` transition can continue generating new partitions until the last vector $a = [0, ..., 0, d]$.

Cycle Continuation

The cycle of generating partitions, computing integrals, and counting results continues until no more partitions are available to process. The Petri net is designed to provide a "head start" by initializing the `counter` place with a default value, enabling the `next_partition` transition to fire before the `feynman_integral` transition completes its first computation.

To extract the result, we compare the precomputed total number of partitions, given by:

$$\text{Total} = \binom{n+d-1}{d},$$

with the number of branch types counted in the place `nb_vectors`. We transfer the final sum to the `output` place.

Partitioning and Synchronization Algorithm

The algorithm `next_partitions` generates a list of m vectors, each representing branch types denoted as a. After computing these m branch types, the last one (i.e., the m-th branch) is saved as b. This process ensures that partitioning is manageable and synchronized with the `FeynmanIntegral` computation.

Algorithm 9: next_partitions

Input: Branch type a, integer m
Output: List v of m branch types, branch type b

```
Let n = size(a);
Let d = sum(a);
Initialize empty list v and empty branch type b;
for i = 0 to m − 1 do
    if a[n − 1] = d then
        break;
    else
        a_i ← next_partition(a);
        Add a_i to v;
if a_i[n − 1] ≠ d then
    Set b = a_i;
```

Control Flow

After partitioning each set of m branch types, the function `FeynmanIntegral` operates on the last branch type b. The control flow is such that:

- `next_partitions` generates a list of m branch types and stores the last one as b.
- Once the m-th branch type is computed and stored in b, the algorithm pauses to call the `FeynmanIntegral` function on b.
- The system waits for `FeynmanIntegral` to complete its computation on b before generating the next set of m branch types.

This control flow ensures that the partitioning process does not overwhelm the evaluation of the Feynman integrals. It coordinates the generation of partitions with the integral Integral, so that only m vectors are produced before the next computation step.

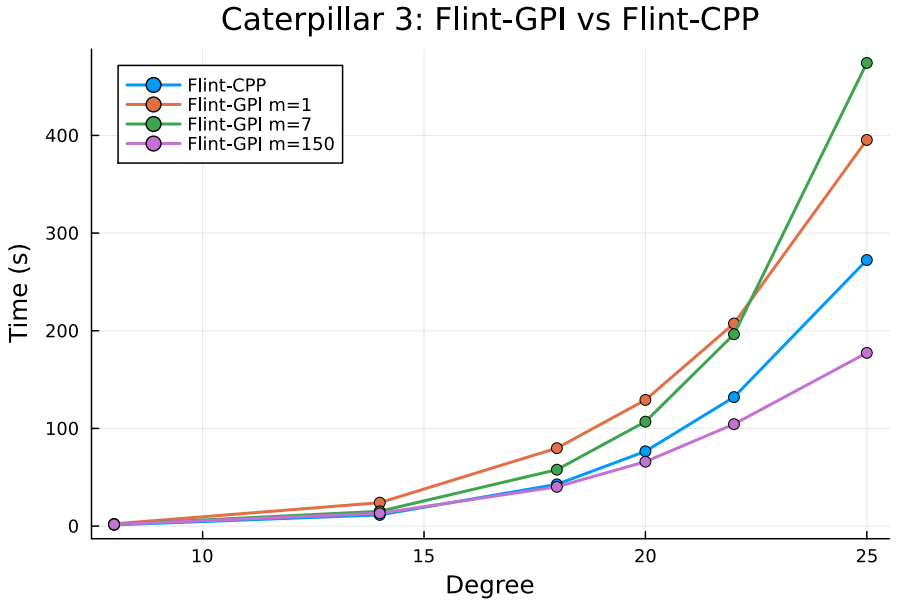
Synchronization

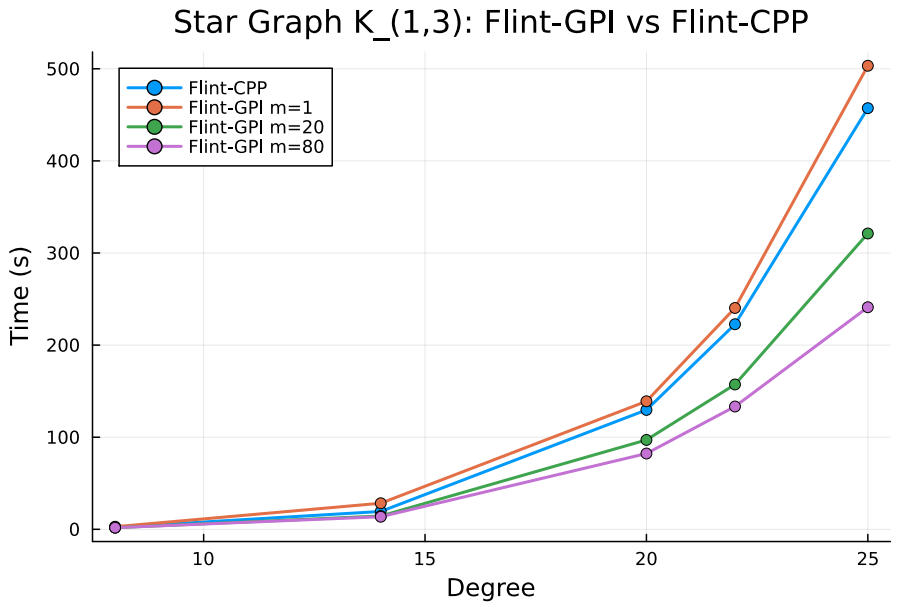
To ensure proper synchronization, the algorithm imposes a constraint that the function `next_partitions` must wait for the `FeynmanIntegral` computation to complete on *branch* before it generates the next set of m branch types. This prevents an overflow of branch types by keeping the number of vectors manageable and avoiding overwhelming the system with too many branches before they are processed.

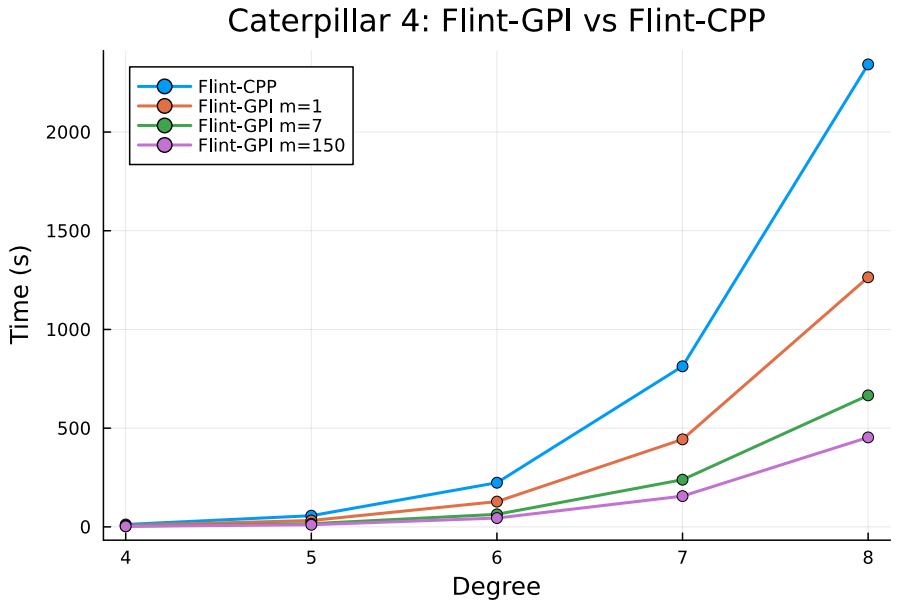
4.4 Timing

We will compare our parallel implementation algorithm in GPI-Space with the sequential C++ version of the code. We observe that as the computation time of the `feynman_integral` function increases, the GPI-Space implementation becomes faster. This is particularly evident when the graph has more edges (i.e., the length of the vector a is longer) or when the degree is higher.

Example 4.4.1. *Here, for a fixed graph, we compare the timing of our algorithm implemented in C++ and GPI-Space as a function of the number of cores.*







4.5 Quasimodularity in Three Settings and Algorithmic Computation

In this section, we will present three cases in which quasimodularity arises from enumerative geometry problems involving elliptic curves. We then provide an explicit algorithm for expressing each *Feynman integral* as a quasimodular form in the generators E_2, E_4, E_6.

4.5.1 Quasimodular Forms and Eisenstein Series

Following [42], a function

$$f : \mathcal{H} \to \mathbb{C}$$

on the complex upper half-plane $\mathcal{H}$ is said to be *almost holomorphic modular of weight* k if

$$f\left(\tfrac{a\tau+b}{c\tau+d}\right) = (c\tau+d)^k f(\tau) \quad \text{for all} \quad \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{SL}_2(\mathbb{Z}),$$

and if it admits a Laurent expansion $f(\tau) = \sum_{j=0}^{r} \frac{f_j(\tau)}{v^j}$ (where $\tau = u + iv$) whose coefficients are holomorphic in $\mathbb{H} \cup \{\infty\}$. The *constant term* in that expansion is a *quasimodular form*, and the largest power of $\frac{1}{v}$ that appears is called the *depth* of the quasimodular form. Equivalently, quasimodular forms are precisely those elements of the polynomial ring generated by the three Eisenstein series

$$E_2(\tau), \quad E_4(\tau), \quad E_6(\tau).$$

A quasimodular form is said to be *of weight* w if it is a homogeneous polynomial of total degree w in E_2, E_4, E_6. A *mixed-weight* quasimodular form is a non-homogeneous polynomial in these Eisenstein series.

4.5.2 Case 1: Gromov–Witten/Hurwitz Invariants of an Elliptic Curve

(See the paper [17] for more details). A fundamental case of quasimodularity occurs with *Hurwitz numbers* counting ramified covers of the elliptic curve (torus). Let

$$F(q) \;=\; \sum_{d} N_{d,g}\, q^{2d}$$

as defined in Definition 2.2.1. By work of Dijkgraaf, Kaneko–Zagier, and others [16, 42], it is known that $F(q)$ is a quasimodular form in E_2, E_4, E_6 of weight $6g - 6$. More refined combinatorial decompositions involve *Feynman graphs* [17]

$$F_g(q) \;=\; \sum_{\Gamma} \frac{1}{|\mathrm{Aut}(\Gamma)|} \sum_{\Omega} I_{\Gamma,\Omega}(q),$$

where the sum is over appropriate Feynman graphs Γ and orders Ω. Theorem 2.6 in [17] ensures that each summand

$$I_{\Gamma,\Omega}(q)$$

is already a mixed-weight quasimodular form, with highest-weight part up to $6g-6$. Their total sum F_g then turns out to be homogeneous of weight $6g-6$ (see [17], Corollary 8.4).

4.5.3 Case 2: Tropical Mirror Symmetry in Dimension One

(see [18] for more details)

In this case, we have the *Gromov–Witten invariants* of an elliptic curve E. For domain curves of genus $g \geq 2$, consider

$$F(q) \;=\; \sum_{d} \Big\langle \tau_{k_1}(\mathrm{pt}) \,\ldots\, \tau_{k_n}(\mathrm{pt}) \Big\rangle_{g,n}^{E,d} q^d,$$

where the notation $\langle \cdot \rangle_{g,n}^{E,d}$ refers to Gromov–Witten invariants in degree d. This series turns out to coincide with certain *Hurwitz counts* (torus covers) under specific ramification conditions, and hence inherits quasimodularity [16, 42].

Moreover, one can write

$$F(q) \;=\; \sum_{\Gamma} \frac{1}{|\mathrm{Aut}(\Gamma)|} \sum_{\Omega} I_{\Gamma,\Omega}(q),$$

where Γ ranges over suitable Feynman graphs and Ω over possible vertex orderings. By [42], each piece $I_{\Gamma,\Omega}(q)$ is a quasimodular form of *mixed* weight, with maximum possible weight

$$\mathrm{weight}_{\max} \;=\; 2\Big(r \;+\; \sum_{i=1}^{n} g_i\Big),$$

where r is the number of edges of the underlying tropical graph, and $\sum_{i=1}^{n} g_i$ is the total genus distribution among the vertices. Summing all pieces often cancels lower-weight contributions, leaving $F(q)$ as a homogeneous quasimodular form.

4.5.4 Case 3: Curves in $E \times \mathbb{P}^1$ and Pearl Chains

(see [24] for more details)

Another example arises in enumerating genus-g curves in $E \times \mathbb{P}^1$ of bidegree (d_1, d_2). Oberdieck–Pixton [43] prove quasimodularity statements for cycle-valued Gromov–Witten invariants in this context.

On the tropical side, one obtains *pearl chains* $\mathcal{P}$ that encode combinatorial data for tropical maps to $E_{\mathbb{T}} \times \mathbb{P}^1_{\mathbb{T}}$. Each such chain contributes a sum of Feynman-type integrals

$$\sum_{d_1} N^{\mathcal{P}}_{(d_1,d_2,g)}\, q^{d_1} \;=\; \frac{1}{|\mathrm{Aut}(\mathcal{P})|} \sum_{\Omega} I_{\mathcal{P},\Omega}(q),$$

and each summand $I_{\mathcal{P},\Omega}(q)$ is quasimodular of mixed weight at most $4(d_2 + g - 1)$ (see [42, §6]). Summing over *all* pearl chains $\mathcal{P}$ (and orders Ω) recovers the full generating function for such curves, ensuring overall quasimodularity of weight at most $4(d_2 + g - 1)$. In special cases, one may obtain a homogeneous form after cancellations.

4.5.5 Algorithmic Computation of Quasimodular Forms

While each contribution $I_{\Gamma,\Omega}(q)$ or $I_{\mathcal{P},\Omega}(q)$ might be derived as a formal or numeric power series, we can compute the process of matching it to a polynomial in the Eisenstein series E_2, E_4, E_6. The high-level idea is:

1. Extract the power-series expansion of `Iq` (one of the integrals or sums).
2. Compare it against a truncated basis of possible monomials $E_2^a(q)\, E_4^b(q)\, E_6^c(q)$ such that $2a + 4b + 6c \leq$ `weightmax`.
3. Use linear algebra on the q-expansions to solve for the coefficients of these monomials.
4. Reconstruct the final polynomial in E_2, E_4, E_6.

We now describe several algorithms required to define the relevant quasimodular form:

Algorithm 10: *Counts the number of monomials $E_2^{e_2} E_4^{e_4} E_6^{e_6}$ whose total weight is at most* `weightmax`.

Algorithm 11: *Enumerates all monomials in E_2, E_4, E_6 up to the specified total weight and returns them as a list of symbolic expressions.*

Algorithm 13: *A univariate variant of* `filter_term` *that keeps only the polynomial terms up to a specified degree.*

Algorithm 14: *Applies univariate* `filter_term` *to each polynomial in a vector, returning the filtered vector.*

Algorithm 17: *Main routine. Given a polynomial* `Iq` *and a* `weightmax`*, returns the quasi-modular form in* E_2, E_4, E_6 *that matches* `Iq` *up to the required expansions.*

Algorithm 10: number_monomial

Input: Integer $weightmax$
Output: Integer $count$

```
count ← 0;
w ← {2, 4, 6};
for e2 = 0 to weightmax do
    for e4 = 0 to weightmax do
        for e6 = 0 to weightmax do
            degree ← 2e2 + 4e4 + 6e6;
            if 0 < degree ≤ weightmax then
                Increment count;
return count;
```

Algorithm 11: express_as_eisenstein_series

Input: Integer $weightmax$
Output: List of expressions in (E_2, E_4, E_6)

```
empty list expressions;
for e2 = 0 to weightmax do
    for e4 = 0 to weightmax do
        for e6 = 0 to weightmax do
            d ← 2e2 + 4e4 + 6e6;
            if 0 < d ≤ weightmax then
                term ← 1;
                if e2 ≠ 0 then
                    term ← term × E2^e2;
                if e4 ≠ 0 then
                    term ← term × E4^e4;
                if e6 ≠ 0 then
                    term ← term × E6^e6;
                Add term to expressions;
return expressions;
```

Algorithm 12: filter_term (multivariate)

Input: Polynomials $pols$, Variables $variables$, Powers $power$; `// powers are the powers of the variables`

Output: Filtered polynomial $result$; `// Filtered polynomial is the sum of the terms of` *pols* `with degree less than or equal to` *power*

```
T ← parent(variables);
gensR ← gens(T);
position ← findfirst(var → var == variables, gensR);
result ← 0;
d ← Vector of length length(variables);
for each term in terms(T(pols)) do
    for j = 1 to length(variables) do
        po ← position[j];
        d[j] ← degree(term, po);
    if all(d ≤ power) then
        result += result + term;
return result;
```

Algorithm 13: filter_term (univariate)

Input: Univariate polynomial $poly$, integer max_degree

Output: Filtered polynomial $result$

```
result ← 0;
for i = 0 to degree(poly) do
    if i ≤ max_degree then
        result ← result + term_i(poly)
return result;
```

Algorithm 14: filter_vector

Input: Vector of polynomials $polyvector$, integer max_degree

Output: Vector of filtered polynomials $result$

```
for each poly in polyvector do
    filter_term(poly, max_degree);
    Add to result;
return result;
```

Algorithm 15: express_as_powers

Input: Integer max_degree, Integer $weightmax$
Output: Vector of polynomials $result$

Initialize empty list $result$;
Set $nb \leftarrow max\_degree$;
Set $E_i \leftarrow$ eisenstein_series(nb, i);
for $e_2 = 0$ **to** $weightmax$ **do**
    for $e_4 = 0$ **to** $weightmax$ **do**
        for $e_6 = 0$ **to** $weightmax$ **do**
            Set $degree \leftarrow 2e_2 + 4e_4 + 6e_6$;
            if $0 < degree \leq weightmax$ **then**
                Add $(E_2^{e_2} E_4^{e_4} E_6^{e_6})$ to $result$;
return $result$;

Algorithm 16: quasi_matrix

Input: Polynomial Iq, Integer $weightmax$
Output: Matrix $B = A \times Q$

Set $m \leftarrow$ degree(Iq);
Set $Evector \leftarrow$ filter_vector(express_as_powers$(m, weightmax), q, m)$;
Set $A \leftarrow$ get_coeff$(Evector)$;
Set $Q \leftarrow$ get_coeff(Iq);
return $B \leftarrow A \times Q$;

Algorithm 17: quasimodular_form

Input: Polynomial Iq, Integer $weightmax$
Output: Quasimodular form p in terms of Eisenstein series E_i

Set $max\_degree \leftarrow$ total_degree(Iq);
Set Vector $b \leftarrow$ quasi_matrix$(Iq, weightmax)$;
Set $comb\_result \leftarrow$ express_as_eisenstein_series$(weightmax)$;
Set $p \leftarrow 0$;
for $(i, term) \in$ *enumerate*$(comb\_result)$ **do**
    if $b_i = 0$ **then**
        continue
    Set $tmp \leftarrow b_i \times term$;
    Set $p \leftarrow p + tmp$;
return (fac, p);

Implementation Remarks

- *The* `quasi_matrix` *function.* In practice, `Iq` is a polynomial in q. We compare it to precomputed expansions of $E_2^a E_4^b E_6^c$ to find the coefficient vector `b`.
- *The* `express_as_eisenstein_series` *function.* This enumerates all possible monomials $E_2^a E_4^b E_6^c$ with $2a + 4b + 6c \leq$ `weightmax`. We store them in a list or dictionary so that the index i matches up with $\mathtt{b}[i]$.
- *Mixed vs. homogeneous weight.* Although each individual summand (e.g. $I_{\Gamma,\Omega}(q)$) might be a *mixed-weight* form, summing them over all relevant graphs/orders can cancel lower weight terms, leaving a homogeneous result.
- *Return value.* The pair $(\mathtt{fac}, \mathtt{p})$ can store additional normalizing factors in `fac`, if needed. The main component `p` is the final quasimodular polynomial in E_2, E_4, E_6.

We have implemented the quasimodular form computation in parallel using the GPI-Space framework. First, we compute the Feynman integral sum, denoted I_q, up to a chosen degree d. Next, we apply the function `quasimodular_form`(I_q, *weightmax*) to express this sum in terms of the Eisenstein series E_2, E_4, E_6. This parallel approach accommodates both the trivalent Feynman-graph (for Hurwitz numbers), the more general Feynman-graph setting (including psi-classes for Gromov–Witten invariants), and the pearl-chain.
In Figure 4.5, we present the Petri net corresponding to the most general case. Variants of this net can be obtained by omitting certain components, such as genus or loop structures, to accommodate specialized cases like trivalent or pearl-chain configurations. The diagram in Figure 4.5 specifically illustrates the workflow designed for the computation of Gromov–Witten invariants.

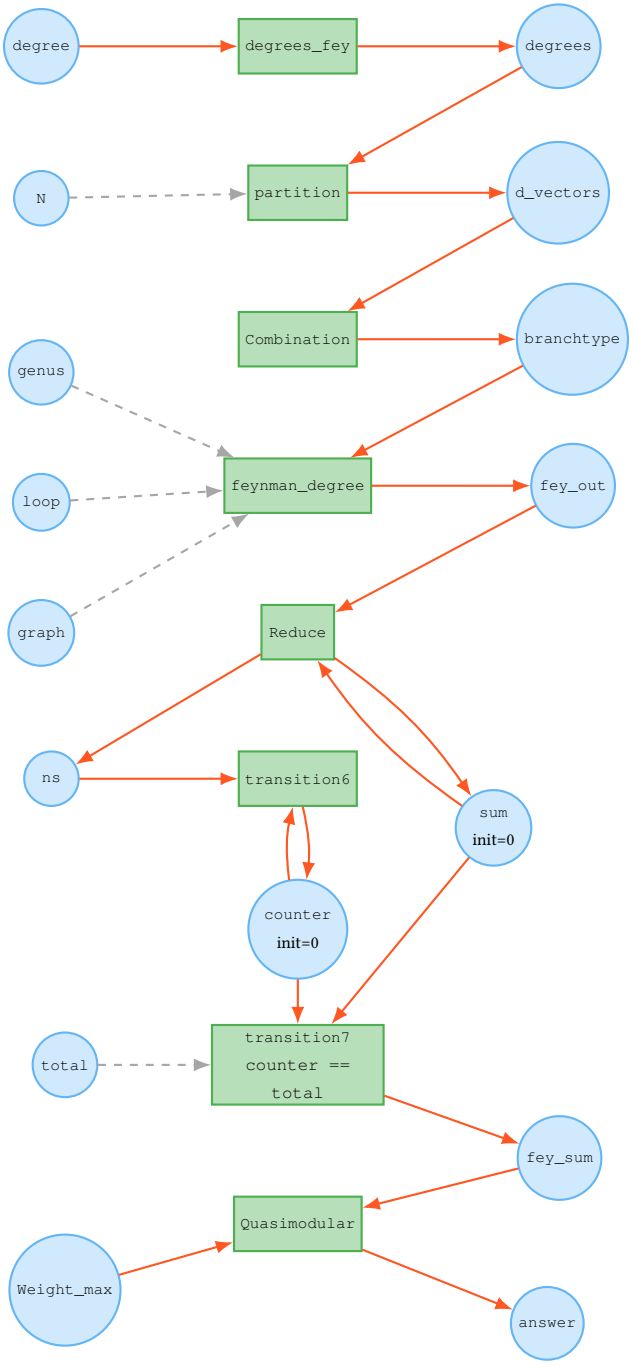


FIGURE 4.5: Computation of Quasimodular Forms using Petri Net

Chapter 5

Vanishing Theorems for Gromov-Witten Invariants

In this chapter, we prove that the Feynman Integral vanishes under specific branch types and given conditions. Furthermore, we introduce new approaches to improve our algorithm for Feynman graphs with multiple edges. We observe that the computation time can be reduced by half by identifying two symmetric vectors (obtained by swapping 0 and -1) in the signature and multiplicities. Additionally, we provide a new proof of the theorem established in [17], which states that the Feynman Integral is zero if and only if the graph contains a bridge.

5.1 Introduction

Mirror symmetry is a concept from string theory that links two areas of geometry: algebraic geometry and symplectic geometry. It has grown into a key idea in modern mathematics. Our work builds on the fact that the generating functions of Gromov–Witten invariants (which count certain curves in a space) are connected to specific integrals on its "mirror space". For elliptic curves, this connection is well-studied, particularly in relation to Hurwitz numbers and Feynman integrals [17]. Exploring this relationship leads to new discoveries in generating functions, (quasi-)modular properties, and how Gromov–Witten invariants can be understood using combinatorial methods.

This chapter aims to speed up the computation of Feynman integrals. These integrals are important because they help us deduce Gromov–Witten invariants via mirror symmetry. We

focus on the case of elliptic curves. The Gromov–Witten theory helps us count the number of branched covers of an elliptic curve with specific branching conditions.

We use tropical geometry as a bridge to connect classical Gromov–Witten theory with Feynman integrals. This approach builds on the Gross–Siebert program, which was developed to describe mirror symmetry. In the context of elliptic curves, tropical mirror symmetry establishes a relationship between Hurwitz numbers, descendant Gromov-Witten invariants, and Feynman integrals. The work of Okounkov and Pandharipande [23] demonstrates that Hurwitz numbers can be viewed as a special case of descendant Gromov-Witten invariants. A more general tropical mirror symmetry theorem for elliptic curves, extending to descendant Gromov-Witten invariants, was later developed in [18].

One main result of this work is to provide a condition for a given Feynman graph in which the Feynman integral vanishes in the trivalent case. We also generalize this result to graphs with more than one connection (tropical psi-classes). Additionally, we establish a bound for the infinite S-function S(z) series in the Feynman integral, beyond which the Feynman integral is zero.

Definition 5.1.1. Let $x_1, \ldots, x_n$ be the vertices of a graph Γ, and let $\underline{a} = (a_1, \ldots, a_r, \ldots, a_m)$ be the branch type. For each vertex x_k, the incident edge weights are:

$$W(x_k) \;=\; \Big[a_{k_1},\, \ldots,\, a_{k_r}\Big],$$

where $a_{k_1}, \ldots, a_{k_r}$ are the branch types associated with the $r = \mathrm{val}(x_k)$ edges incident to x_k.

5.1.1 Graph Setup and Propagator Terms

Consider a graph $\Gamma = (V, E)$ with vertices $V = \{x_1, \ldots, x_n\}$ and edges $E = \{q_1, \ldots, q_r\}$. Each edge $q_k = (x_{k_1}, x_{k_2})$ connects a source vertex $\mathrm{src}(q_k) = x_{k_1}$ to a destination vertex $\mathrm{dst}(q_k) = x_{k_2}$. A branch-type vector $a = (a_1, \ldots, a_r)$ is assigned to the edges, with total sum $d = \sum_{k=1}^{r} a_k$. For each vertex x_i, the valence of x_i is $\mathrm{val}(x_i) = r_i$, and the associated weights are $W_{x_i} = [a_{i_1}, \ldots, a_{i_{r_i}}]$, where $a_{i_j} = 0$ for $j = 1, \ldots, l_i$ and $a_{i_j} > 0$ for $l_i < j \leq r_i$.

The propagator P is constructed from terms depending on a_k:

- **Constant term** ($a_k = 0$):

$$\mathrm{constterm}(x_{k_1}, x_{k_2}, d) = \sum_{w=1}^{d} w\, x_{k_1}^{d+w}\, x_{k_2}^{d-w}.$$

- **Non-constant term** ($a_k > 0$):

$$\text{nonconstterm}(x_{k_1}, x_{k_2}, q_k, a_k, d) = \sum_{w|a_k} w\, x_{k_1}^d\, x_{k_2}^d \left(x_{k_1}^w x_{k_2}^{-w} + x_{k_1}^{-w} x_{k_2}^w\right) q_k^{2a_k}.$$

The total propagator is:

$$P = \prod_{k=1}^{r} \left[\text{propagator term from } q_k\right],$$

expanding as:

$$P = x_1^{r_1 d + l_1} \cdots x_n^{r_n d + l_n} \sum \left[\prod_{i=1}^{n} C_i\, x_i^{\pm t_{i,1} \pm \cdots \pm t_{i,l_i} \pm w_{i,l_i+1} \pm \cdots \pm w_{i,r_i}}\right]$$

where $t_{i,j} \in \{1, \ldots, d\}$ are variables for $a_k = 0$, $w_{i,j} \mid a_k$ are fixed exponents for $a_k > 0$, and C_i are coefficients.

5.1.2 Feynman Integral Definition

For a graph Γ with an ordered set of vertices $\{x_1, \ldots, x_n\}$ under an ordering Ω and edge labels $q_1, \ldots, q_r$, the Feynman integral $I_{\Gamma,\Omega}^{l_1,\ldots,l_n}(q)$ is defined as:

$$I_{\Gamma,\Omega}^{l_1,\ldots,l_n}(q) = \text{Coef}\left[x_1^{l_1} \cdots x_n^{l_n}\right] \left[\prod_{k=1}^{r} P\left(\frac{x_{k_1}}{x_{k_2}}, q_k\right)\right],$$

where x_{k_1} and x_{k_2} are the source and destination vertices of the k-th edge, respectively, and P is the propagator, a function that depends on the vertex ratio $\frac{x_{k_1}}{x_{k_2}}$ and the edge label q_k.

Alternatively, using a specific form of the propagator, the Feynman integral can be expressed as:

$$I_{\Gamma,\Omega}^{l_1,\ldots,l_n}(q) = \text{Coef}\left[x_1^{\text{val}(x_1)d + l_1} \cdots x_n^{\text{val}(x_n)d + l_n}\right] P,$$

where $\text{val}(x_i)$ is the valence (degree) of vertex x_i, d is a parameter, often representing a degree or dimension, and the propagator P is given by:

$$P = \prod_{a_k = 0} \text{constterm}(x_{k_1}, x_{k_2}, d) \cdot \prod_{a_k \neq 0} \text{nonconstterm}(x_{k_1}, x_{k_2}, q_k, a_k, d), \tag{5.1}$$

with $\mathrm{constterm}$ and $\mathrm{nonconstterm}$ being specific functions determined by the graph's structure and edge properties.

Special Case: When $l_1 = l_2 = \cdots = l_n = 0$, the integral simplifies to $I_{\Gamma,\Omega}(q)$, a form studied in [17].

5.1.3 Trivalent Graph Case

For a trivalent graph Γ, where each vertex x_i has valence $\mathrm{val}(x_i) = 3$, the Feynman integral specializes to:

$$I_{\Gamma,\Omega}(q) = \mathrm{Coef}\left[x_1^{3d} \cdots x_{2g-2}^{3d}\right] P,$$

The propagator P is as defined above. The number of vertices is $2g - 2$, consistent with a trivalent graph of genus g.

This particular form occurs naturally when computing tropical Hurwitz numbers and Gromov-Witten invariants in tropical geometry, as detailed in Chapter 2.

5.1.4 Pearl Chain Graphs

Pearl chain graphs arise in the context of curve-counting in $E \times \mathbb{P}^1$, connecting Gromov-Witten invariants to Feynman integrals via tropical stable maps, as explored by Böhm, Goldner, and Markwig in [24].

Definition 5.1.2 (Pearl Chain)**.** A pearl chain $\mathcal{P}$ of type (d_2, g) is a connected bipartite graph with d_2 the number of white vertices of valence ≥ 1 and $d_2 + g - 1$ the number of black vertices, each of valence 2. The edges connect only white and black vertices, and there are no 2-cycles.

These graphs model stable maps in the tropical cylinder $E_T \times \mathbb{P}^1_T$.

The Feynman integral for a pearl chain $\mathcal{P}$ is:

$$I^{l_1,\ldots,l_n}_{\mathcal{P},\Omega}(q) = \mathrm{Coef}\left[x_1^{\mathrm{val}(x_1)d+l_1} \cdots x_n^{\mathrm{val}(x_n)d+l_n}\right] P,$$

where P is the propagator as in 5.1 to the bipartite structure of $\mathcal{P}$.

5.1.5 Algorithmic Improvement via Exponent Systems

To compute $I_{\Gamma,\Omega}^{l_1,\ldots,l_n}(q)$, we extract the coefficient of $x_1^{\mathrm{val}(x_1)d+l_1}\cdots x_n^{\mathrm{val}(x_n)d+l_n}$ from the propagator P. In the expansion of P, terms take the form:

$$x_1^{\mathrm{val}(x_1)d+l_1+e_1}\cdots x_n^{\mathrm{val}(x_n)d+l_n+e_n},$$

where the exponent deviations e_i are:

$$e_i = \pm t_{i,1} \pm \cdots \pm t_{i,l_i} \pm w_{i,l_i+1} \pm \cdots \pm w_{i,r_i},$$

with $t_{i,j} \in \{1,\ldots,d\}$ for edges where $a_k = 0$ and $w_{i,j} \mid a_k$ for edges where $a_k > 0$.

For a term to contribute to the integral, we must have:

$$e_i = 0 \quad \text{for each } i,$$

yielding the system of exponent equations for each vertex x_i :

$$\sum_{\substack{j\sim i\\ a_k=0}} s_{ij}t_{ij} + \sum_{\substack{j\sim i\\ a_k>0}} s'_{ij}w_{ij} = 0 \tag{S}$$

where the s_{ij} and s'_{ij} are ± 1 based on the flip signature f's orientation.

Each variable $t_{i,j}$ or $w_{i,j}$ appears in exactly two equations (corresponding to the source and destination vertices of an edge) with opposite signs. The integral is then:

$$I_{\Gamma,\Omega}^{l_1,\ldots,l_n}(q) = \sum_s C_s,$$

where s indexes the solutions to (S), and C_s are the coefficients of the corresponding monomials. If (S) has no solutions, the integral vanishes: $I = 0$.

5.2 Global Flip Symmetry Theorem

Theorem 5.2.1 (Global Flip Symmetry)**.** *Let $\Gamma = (V, E)$ be a graph with branch type $a = (a_k)$, and let $S(\Gamma, a)$ denote the set of flip signatures. If two flip signatures $f, f' \in S(\Gamma, a)$ differ* only *by swapping -1 and 0 simultaneously across all edges with $a_k = 0$, then their contributions to the Feynman integral satisfy:*

$$I_{\Gamma,a}(f) = I_{\Gamma,a}(f').$$

Proof. We proceed by analyzing the structure of the Feynman integral.

Homogeneous System Structure

The Feynman integral depends on solving a homogeneous system of exponents for each vertex $x_i \in V$:

$$\sum_{\substack{j\sim i \\ a_k=0}} (\pm t_{ij}) + \sum_{\substack{j\sim i \\ a_k>0}} (\pm w_{ij}) = 0 \quad \text{for all } x_i \in V, \tag{5.2}$$

where t_{ij} are variables associated with edges where $a_k = 0$, with signs $(\pm)$ determined by the flip signature f, and w_{ij} are fixed parameters for edges where $a_k > 0$.

For a flip signature f, this system becomes:

$$\sum_{\substack{j\sim i \\ a_k=0}} s_{ij} t_{ij} + \sum_{\substack{j\sim i \\ a_k>0}} s'_{ij} w_{ij} = 0 \quad \text{for all } x_i \in V,$$

where s_{ij} and s'_{ij} are ± 1 based on f's orientation.

System Transformation under Global Flip

A global flip swaps -1 and 0 for all edges with $a_k = 0$, reversing their orientations. For the flipped signature f', the system S' is:

$$\sum_{\substack{j\sim i \\ a_k=0}} (-s_{ij}) t'_{ij} + \sum_{\substack{j\sim i \\ a_k>0}} s'_{ij} w_{ij} = 0 \quad \text{for all } x_i \in V,$$

since the orientation change negates the signs s_{ij} for $a_k = 0$ edges, while $a_k > 0$ terms remain unchanged.

If (t_{ij}, w_{ij}) solves S, then setting $t'_{ij} = -t_{ij}$ solves S', because:

$$\sum_{\substack{j\sim i \\ a_k=0}} (-s_{ij})(-t_{ij}) + \sum_{\substack{j\sim i \\ a_k>0}} s'_{ij} w_{ij} = \sum_{\substack{j\sim i \\ a_k=0}} s_{ij} t_{ij} + \sum_{\substack{j\sim i \\ a_k>0}} s'_{ij} w_{ij} = 0.$$

This defines a mapping: $(t_{ij}, w_{ij}) \mapsto (-t_{ij}, w_{ij})$.

Solution Space Equivalence.

Since the system is linear and homogeneous, the mapping $(t_{ij}, w_{ij}) \mapsto (-t_{ij}, w_{ij})$ is a bijection between the solution sets:

$$\text{Solution}(S) \;\leftrightarrow\; \text{Solution}(S')$$

preserving the number of solutions:

$$|\text{Solution}(S)| = |\text{Solution}(S')|.$$

However, the Feynman integral sums coefficients over these solutions, not just their count.

The propagator is:

$$P = x_1^{d\cdot r_1} \cdots x_n^{d\cdot r_n} \sum_{(t,w)\in \text{Solution}(S)} C_{(t,w)} x_1^{\alpha_1} \cdots x_n^{\alpha_n},$$

where $\alpha_i = \sum(\pm t_{ij} \pm w_{ij}) = 0$ for solvable terms, simplifying to:

$$P = x_1^{d\cdot r_1} \cdots x_n^{d\cdot r_n} \sum_{(t,w)\in \text{Solution}(S)} C_{(t,w)}.$$

The Feynman integral $I_{\Gamma,a}(f)$ is the coefficient of $x_1^{d\cdot r_1} \cdots x_n^{d\cdot r_n}$:

$$I_{\Gamma,a}(f) = \sum_{(t,w)\in \text{Solution}(S)} C_{(t,w)}.$$

For the flipped signature f', we have:

$$I_{\Gamma,a}(f') = \sum_{(t',w)\in \text{Solution}(S')} C_{(t',w)}.$$

Using the bijection $(t', w) = (-t, w)$, we compare:

$$I_{\Gamma,a}(f) = \sum_{(t,w)\in \text{Solution}(S)} C_{(t,w)}, \quad I_{\Gamma,a}(f') = \sum_{(-t,w)\in \text{Solution}(S')} C_{(-t,w)}.$$

Since $C_{(t,w)} = C_{(-t,w)}$, the sums are equal:

$$I_{\Gamma,a}(f) = \sum_{(t,w)\in \text{Solution}(S)} C_{(t,w)} = \sum_{(-t,w)\in \text{Solution}(S')} C_{(-t,w)} = I_{\Gamma,a}(f').$$

□

Remark 5.2.2 (Partial vs. Global Flips)**.** A global flip $-1 \leftrightarrow 0$ works *only* if it is applied to *every* zero-weight edge at once. Flipping a strict subset of such edges disrupts the uniform sign negation and typically produces a different exponent system (S'), hence a different integral.

Practical Implication: Swapping $-1 \leftrightarrow 0$ in any signature keeps its Feynman contribution the same. Consequently, if two signatures differ *only* by this swap, one can safely remove one and double the other's multiplicity. This reduces both the number of signature vectors stored and the final integral's running time by almost half (see the table 5.1).

Algorithmic.

Example 5.2.3. *Consider*

```
julia> signature_and_multiplicities(G, a)
8-element Vector{Tuple{Int64, Vector{Int64}}}:
(5, [-1, 1, 2, 0, 0, 1])
(1, [0, 1, 2, -1, 0, 1])
(3, [0, 1, 2, 0, -1, 1])
(3, [-1, 1, 2, -1, 0, 1])
(1, [-1, 1, 2, 0, -1, 1])
(5, [0, 1, 2, -1, -1, 1])
(3, [-1, 1, 2, -1, -1, 1])
(3, [0, 1, 2, 0, 0, 1])
```

In this implementation, we identify vectors that differ only by the swap $-1 \leftrightarrow 0$, merge them, and sum their multiplicities. The final output is half the size:

```
julia> signature_and_multiplicities(G, a)
4-element Vector{Tuple{Int64, Vector{Int64}}}:
(10, [-1, 1, 2, 0, 0, 1])
(2, [0, 1, 2, -1, 0, 1])
```

Algorithm 18: `signature_and_multiplicities`

Input: Graph Γ, branch type $\underline{a}$.

Output: List of flip signatures (vectors) with associated multiplicities.

begin
  $E \leftarrow \{\text{edges of } \Gamma\}$ $\qquad (x_{k_1} - e_k - x_{k_2})$;
  $V \leftarrow \{\text{vertices } x_{k_i} \mid e_k \in E \text{ and } a_k = 0\}$;
  $P \leftarrow S(V)$; `// all permutations of` V
  $D \leftarrow \{\}$; `// (empty dictionary)`
  $\mathrm{nv}(\Gamma) \leftarrow$ number of vertices of Γ
  $\mathrm{nbzero}(a) \leftarrow$ number of zero-weight edges of Γ
  if $\mathrm{nbzero}(a) \leq 1$ **then**
    $D \leftarrow \left(\mathrm{nv}(\Gamma)!, \underline{a}\right)$
  else
    foreach $\Omega \in P$ **do**
      $y \leftarrow \mathrm{sgn}(\Gamma, \Omega, \underline{a})$;
      if $y \in D$ **then**
        $D[y] \leftarrow D[y] + 1$
      else
        $D[y] \leftarrow 1$
    foreach $y \in D$ **do**
      $D[y] \leftarrow D[y] \times \dfrac{(\mathrm{nv}(\Gamma))!}{|P|}$
    foreach *vector* a *in* D **do**
      foreach *vector* b *in* D **do**
        if `swapping` -1 *and* 0 *in* a *produces* b **then**
          Remove b from D;
          $D[a] \leftarrow D[a] \times 2$;
  return D

```
(6, [0, 1, 2, 0, -1, 1])
(6, [-1, 1, 2, -1, -1, 1])
```

5.2.1 Repeated Edges and Permutations

We now consider the simplifications possible when dealing with graphs that have repeated edges connecting the same pair of vertices.

Proposition 5.2.4 (Repeated Edge Permutation). *Let $\Gamma = (V, E)$ be a graph in which some edges $q_{i_1}, \ldots, q_{i_s}$ connect the same vertices (x_{k_1}, x_{k_2}). Suppose the branch-type vector a assigns weights $a_{i_1}, \ldots, a_{i_s}$ to these repeated edges. Then, permuting the entries $(a_{i_1}, \ldots, a_{i_s})$ yields the same Feynman integral, up to a relabeling of factors $q_{i_k}^{2\,a_{i_k}}$.*

Sketch of Proof. Repeated edges connecting the same two vertices are structurally indistinguishable, differing only in their assigned weights a_{i_j}. Permuting these weights corresponds to a simple relabeling of identical edges. Since the integral computation is symmetric in edge labels, the integral itself remains unchanged. The only difference is the labeling of the edge-variables $q_{i_k}^{2\,a_{i_k}}$, which does not affect the computed integral value. □

Computations with Repeated Edges.

To efficiently handle repeated edges, we adopt the following approach:

1. Select a single representative arrangement of the branch-type weights.
2. Compute the integral only once for this arrangement.
3. Multiply the resulting integral by the number of distinct permutations of the repeated edge weights.

This eliminates redundant computations, significantly improving computational efficiency.

Example 5.2.5. *Consider the* Caterpillar-4 graph *shown in Figure 5.1.*

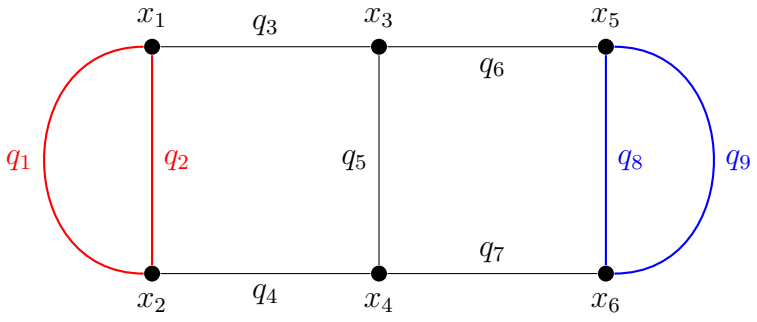


FIGURE 5.1: Caterpillar genus 4

This graph contains two pairs of repeated edges:

$$q_1 = q_2 = (x_1, x_2), \quad q_8 = q_9 = (x_5, x_6).$$

Consider a branch-type vector:

$$a = [4,\ 0,\ 1,\ 5,\ 7,\ 3,\ 0,\ 4,\ 2].$$

We have:

- *Edge weights for the pair* (q_1, q_2)*:* $(4, 0)$.
- *Edge weights for the pair* (q_8, q_9)*:* $(4, 2)$.

By Proposition 5.2.4, permuting these weights (e.g., swapping the order to $(0, 4)$ *or* $(2, 4)$*) will yield the same integral value, up to a relabeling of the edges. Explicit computational verification confirms this symmetry:*

```
    # Original a
julia> a = [1, 2, 0, 1, 0, 0, 0, 2, 0]
julia> feynman_integral_branch_type(F, a)
192 * q[1]^2 * q[2]^4 * q[4]^2 * q[8]^4

# Swapping q8 and q9
julia> a = [1, 2, 0, 1, 0, 0, 0, 0, 2];
julia> feynman_integral_branch_type(F, a)
192 * q[1]^2 * q[2]^4 * q[4]^2 * q[9]^4

# Swapping q1 and q2
julia> a = [2, 1, 0, 1, 0, 0, 0, 0, 2];
julia> feynman_integral_branch_type(F, a)
192 * q[1]^4 * q[2]^2 * q[4]^2 * q[9]^4

# Swapping q8 and q9 again
julia> a = [2, 1, 0, 1, 0, 0, 0, 2, 0];
julia> feynman_integral_branch_type(F, a)
192 * q[1]^4 * q[2]^2 * q[4]^2 * q[8]^4.
```

Thus, we only need to compute the integral once per unique arrangement of weights for repeated edges, reducing computational complexity.

In conclusion, Proposition 5.2.4 ensures significant efficiency gains by the use of symmetry to reduce redundant computations.

5.3 Vanishing Criteria for the General Feynman Integral

Let P be a Feynman graph with vertex valences $\mathrm{val}(x_1), \ldots, \mathrm{val}(x_n)$ and branch type $\underline{a} = (a_1, \ldots, a_r, \ldots, a_n)$. Assume the total degree is $d = \sum_i a_i$. Let an ordering Ω of the vertices and an exponent vector $(l_1, \ldots, l_n) \in \mathbb{Z}^n$ be given.

Theorem 5.3.1 (Vanishing Conditions). *Let x_i be a vertex of the graph P, and let*

$$\underline{a} = (a_1, \ldots, a_r)$$

list the non-constant edge weights incident to x_i. The Feynman integral $I_{P,\Omega}^{(l_1,\ldots,l_n)}(a, q)$ vanishes if either of the following conditions is satisfied:

(1) $|l_i| > \sum_{j=1}^{\mathrm{val}(x_i)} a_j$, *where* $\mathrm{val}(x_i)$ *counts the edges incident to* x_i.

(2) All edge weights a_j are odd, but l_i and $\mathrm{val}(x_i)$ have different parity.

Proof. The Feynman integral is given by extracting the coefficient of the monomial

$$x_1^{\mathrm{val}(x_1)d+l_1} \cdots x_n^{\mathrm{val}(x_n)d+l_n}$$

in the expanded propagator product. Consider the vertex x_i. Each edge with weight $a_j > 0$ incident to x_i contributes an exponent of the form $\pm w_j$, where $w_j \mid a_j$. Thus, to match the required exponent at vertex x_i, we must have

$$\sum_{j=1}^{\mathrm{val}(x_i)} (\pm w_j) = l_i.$$

Condition (1) Exponent Bound.
Assume that

$$|l_i| \; > \; \sum_{j=1}^{\mathrm{val}(x_i)} a_j.$$

Since each divisor $w_j \leq a_j$, the absolute value of the sum satisfies

$$\left| \sum_{j=1}^{\mathrm{val}(x_i)} (\pm w_j) \right| \; \leq \; \sum_{j=1}^{\mathrm{val}(x_i)} a_j.$$

Thus, if $|l_i|$ exceeds this bound, the required exponent cannot appear, and the integral is zero.

Condition (2) Parity Argument.
Assume that all edge weights a_j are odd. Then all divisors $w_j \mid a_j$ are also odd. The sum of $\mathrm{val}(x_i)$ signed odd numbers:

$$\pm w_1 \pm w_2 \pm \cdots \pm w_{\mathrm{val}(x_i)}$$

is even if $\mathrm{val}(x_i)$ is even, and odd if $\mathrm{val}(x_i)$ is odd. Thus, the parity of the exponent shift is exactly the parity of $\mathrm{val}(x_i)$. If the parity of l_i differs from this parity, no solution exists, and hence the integral vanishes. □

Corollary 5.3.2 (Trivalent Graphs with Odd Weights)**.** *Let Γ be a trivalent graph (each vertex has exactly three edges). If a vertex x_i has incident edge weights*

$$W(x_i) = [a_{k_1}, a_{k_2}, a_{k_3}]$$

that are all positive and odd, then the Feynman integral vanishes:

$$I_{\Gamma,\Omega}(a, q) = 0.$$

Proof. This result follows directly from condition (2) of Theorem 5.3.1. Indeed, since all edge weights are odd, the parity argument in the theorem applies. Specifically, the sum of three signed odd numbers is always odd, never zero, preventing the necessary exponent from appearing. Thus, by Theorem 5.3.1, the integral must vanish. □

Example 5.3.3. *Consider the Caterpillar-4 graph shown in Figure 5.1 with the branch type:*

$$a = [4,\ 0,\ 1,\ 5,\ 7,\ 3,\ 0,\ 4,\ 2].$$

The local weights at each vertex are summarized below:

Vertex x_i	$W(x_i)$
x_1	$[4, 0, 1]$
x_2	$[4, 0, 5]$
x_3	$[1, 7, 3]$
x_4	$[5, 7, 0]$
x_5	$[3, 4, 2]$
x_6	$[0, 4, 2]$

Observe that vertex x_3 has exactly three incident edges, each with an odd *weight $(1, 7, 3)$. By condition (2) of Theorem 5.3.1, the integral must vanish, since no combination of three odd integers can sum to zero. Indeed, we explicitly confirm computationally that:*

$$I_{\Gamma,\Omega}(a, q) = 0.$$

The computational verification using JULIA *is shown below:*

```
julia> ve=[(1,2),(1,2),(1,3),(2,4),(3,4),(3,5),(4,6),(5,6),(5,6)]
julia> F = FeynmanIntegral(ve);

julia> a = [4,0,1,5,7,3,0,4,2]
julia> feynman_integral_branch_type(F, a)
0
```

This result is consistent with Theorem 5.3.1.

5.3.1 Removing Vanishing Signatures

We now present a generalized condition under which certain flip signatures always yield zero contributions, enabling their removal to improve computational efficiency.

Proposition 5.3.4 (Vanishing Condition)**.** *Let x_i be a vertex of the graph Γ incident only to edges of zero weight, i.e.,*

$$W(x_i) = [0, 0, \ldots, 0] \quad \textit{with} \quad \mathrm{val}(x_i) \geq 2.$$

If vertex x_i acts consistently either as the source or as the destination for all its incident edges, then any flip signature assigning identical labels $(0, \ldots, 0)$ or $(-1, \ldots, -1)$ to these edges yields a zero contribution to the local Feynman integral.

Proof. For edges with zero weight incident to x_i, the contribution is given by the local propagator factor:

$$\mathrm{constterm}(x_i, x_j, d) = \sum_{w=1}^{d} w\, x_i^{d+w} x_j^{d-w}.$$

If vertex x_i is consistently a *source*, each incident edge contributes a factor of the form:

$$x_i^{d+w_k} \quad \text{with} \quad w_k > 0.$$

Multiplying these terms for all $\mathrm{val}(x_i)$ edges, we obtain:

$$P = x_i^{\mathrm{val}(x_i)\,d+(w_1+w_2+\cdots+w_{\mathrm{val}(x_i)})}.$$

Since each $w_k > 0$, the sum $w_1 + w_2 + \cdots + w_{\mathrm{val}(x_i)}$ is strictly positive, implying that the exponent is strictly larger than $\mathrm{val}(x_i)d$. Hence, we never obtain exactly the required exponent $x_i^{\mathrm{val}(x_i)d}$. Therefore, the contribution to the Feynman integral is zero.

The same reasoning applies symmetrically if x_i acts consistently as the *destination*, leading to a strictly negative sum and again no exact match of the target exponent.

Thus, signatures assigning either all 0 or all -1 to the zero-weight edges at such vertices vanish and can be removed from the computations. □

Table 5.1: Timing of Feynman Integral Computations (in seconds)

Graph	Degree	Feynman	Doubles Signature	Double Edges+ Signature	Double+Vanishing Theorem
$[(1,3),(1,2),(1,2),$ $(2,4),(3,4),(3,4)]$	4	0.08	0.03	0.11	0.01
	10	5.69	3.09	1.26	1.25
	14	34.51	18.84	8.60	7.75
	18	139.94	78.82	42.46	35.23
	20	255.99	142.31	85.15	69.33
	25	931.74	534.40	468.11	337.61
$[(1,2),(1,3),(1,4),$ $(2,3),(2,4),(3,4)]$	8	3.15	1.60	1.19	1.52
	10	9.78	5.07	5.19	4.81
	14	59.95	30.08	30.57	29.67
	18	239.19	125.34	124.33	119.59
	20	435.32	227.79	226.27	217.63
	25	1550.44	834.97	832.35	790.33
$[(1,6),(1,2),(1,2),$ $(2,3),(3,4),(3,4),$ $(4,5),(5,6),(5,6)]$	2	0.89	0.07	0.26	0.03
	4	16.78	8.62	2.95	3.01
	6	459.25	232.64	67.12	67.41
	7	1634.17	820.15	224.50	220.38
	8			653.12	641.76

5.3.2 Graph with Bridges

In their article [17], the authors prove that a Feynman integral is zero for all branch types if and only if the graph contains a bridge. In the following, we reprove the same theorem using an alternative method.

Theorem 5.3.5 ([17]). *A Feynman graph Γ satisfies $N^{trop}_{\underline{a},\Gamma} = 0$ for every branch type $\underline{a}$ if and only if Γ contains a bridge. Equivalently , the integral*

$$I_\Gamma(q_1, \dots, q_{3g-3}) \;=\; 0$$

for all tuples $(q_1, \dots, q_{3g-3})$ if and only if Γ has a bridge.

Proof. Consider a graph with n vertices and r edges, and let $\underline{a} = (a_1, \dots, a_r)$ be a branch type. Using the decomposition

$$\Big[\prod_{a_k=0} \text{constterm}(\cdots) \;\times\; \prod_{a_k\neq 0} \text{nonconstterm}(\cdots) \Big],$$

we obtain a system of equations depending on whether $a_k = 0$ or $a_k \neq 0$. Specifically:

- If $a_k = 0$, the equation involves a sum of the form $\sum_{w=1}^{d}$.
- If $a_k \neq 0$, the equation involves a sum of the form $\sum_{0<w|a_k}$.

Now, consider a non-oriented graph with a bridge. Let the bridge edge be w_{CD}, connecting two disjoint subgraphs G_1 (containing vertex C) and G_2 (containing vertex D).

Equations at the Bridge Vertices:

- In G_1, the equation at vertex C involves w_{CD}:

$$\pm w_{CD} \pm (\text{other edges at } C) = 0.$$

- In G_2, the equation at vertex D involves w_{CD}:

$$\mp w_{CD} \pm (\text{other edges at } D) = 0.$$

- Summing all equations in G_1, the internal edges cancel out (as they appear twice with opposite signs), leaving:

$$\pm w_{CD} = 0.$$

  This implies $w_{CD} = 0$.

- Similarly, summing all equations in G_2 also forces:

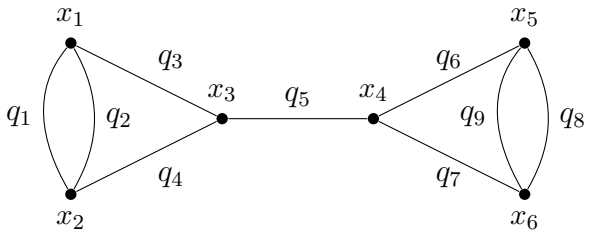
$$\mp w_{CD} = 0,$$

  which again implies $w_{CD} = 0$.

Since $w_{CD} > 0$, we got a contradiction. Conversely, for the opposite direction, we refer to the proof in [17].

□

Example 5.3.6. *Consider the graph Γ with a bridge at edge q_5:*



The weight equations at each vertex impose constraints on the exponents of the monomials in the Feynman integral. The general form of the monomials (up to sign conventions) is:

$$x_1^{-a_{12}+a_{21}+a_{13}} x_2^{a_{12}-a_{21}+a_{23}} x_3^{-a_{13}-a_{23}+a_{34}} x_4^{-a_{34}+a_{45}+a_{46}} x_5^{-a_{45}+a_{56}-a_{65}} x_6^{-a_{46}-a_{56}+a_{65}}.$$

The balancing equations at each vertex are:

$$\begin{cases} -a_{12} + a_{21} + a_{13} = 0, \\ a_{12} - a_{21} + a_{23} = 0, \\ -a_{13} - a_{23} + a_{34} = 0, \\ -a_{34} + a_{45} + a_{46} = 0, \\ -a_{45} + a_{56} - a_{65} = 0, \\ -a_{46} - a_{56} + a_{65} = 0. \end{cases}$$

Summing the last two equations:

$$(-a_{45} + a_{56} - a_{65}) + (-a_{46} - a_{56} + a_{65}) = 0,$$

which simplifies to:

$$-a_{45} - a_{46} = 0 \quad \Rightarrow \quad a_{45} = -a_{46}.$$

Substituting this into the equation at vertex x_4:

$$-a_{34} + a_{45} + a_{46} = 0,$$

and using $a_{45} = -a_{46}$, we obtain:

$$-a_{34} = 0 \quad \Rightarrow \quad a_{34} = 0.$$

This forces the edge q_5 to have zero weight, contradicting the requirement that all edge weights be strictly positive. Thus, the Feynman integral must vanish:

$$I_\Gamma(q_1, \ldots, q_9) = 0.$$

5.3.3 Remark: Invariance Under Permutations for Two-Vertex Graphs

Remark 5.3.7. Let Γ be a graph with exactly two vertices x_1 and x_2. Suppose the branch type is given by $\underline{a} = (a_1, a_2, \ldots, a_r)$, where each a_k represents the weight of an edge between x_1 and x_2. Then, the Feynman integral remains invariant under any permutation of the sequence $\underline{a}$:

$$I_\Gamma(a_1, a_2, \ldots, a_r) = I_\Gamma(\sigma(a_1, a_2, \ldots, a_r))$$

for any permutation σ of the weights.

In fact. The integral depends on monomials of the form

$$x_1^{\mathrm{val}(x_1)d+l_1} x_2^{\mathrm{val}(x_2)d+l_2},$$

where $d = \sum a_i$ is the total degree. Since every edge in Γ connects x_1 and x_2, reordering the weights does not affect the sum $\sum a_j$ or the set of possible divisors $w \mid a_k$ contributing to the expansion. Thus, the Feynman integral remains unchanged under permutations of $\underline{a}$.

Example 5.3.8. *Consider the graph 5.2. The associated Feynman integral at a branch type* $a = (1, 2, 3)$ *is given by*

$$32q_1^2q_2^4q_3^6.$$

For all permutations b *of* $(1, 2, 3)$*, the associated Feynman integral gives the same result up to permutation of* q^a*.*

```
julia> using Combinatorics

julia> ve=[(1,2),(1,2),(1,2)]

julia> G=feynman_graph(ve)
FeynmanGraph([(1, 2), (1, 2), (1, 2)])

julia> a=[1,2,3];

julia> F=FeynmanIntegral(G);

julia> A=unique(permutations(a))
6-element Vector{Vector{Int64}}:
 [1, 2, 3]
 [1, 3, 2]
 [2, 1, 3]
 [2, 3, 1]
 [3, 1, 2]
 [3, 2, 1]

julia> for a in A
           println(feynman_integral_branch_type(F,a))
       end
32*q[1]^2*q[2]^4*q[3]^6
32*q[1]^2*q[2]^6*q[3]^4
32*q[1]^4*q[2]^2*q[3]^6
32*q[1]^4*q[2]^6*q[3]^2
32*q[1]^6*q[2]^2*q[3]^4
32*q[1]^6*q[2]^4*q[3]^2
```

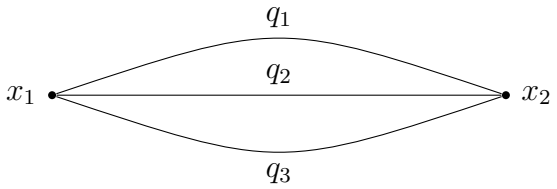


FIGURE 5.2: Caterpillar genus 2.

Remark 5.3.9. The above example does not work in general for graphs with more than two vertices, across all permutations P of a. However, for a subset $S \subset P(a)$ of permutations, the Feynman integral at a branch type a remains invariant up to a permutation of q^a. It would be interesting to determine a general condition on S under which the Feynman integral is invariant up to a permutation of q^a.

Chapter 6

Automated Transformation of the Sector Structure of Feynman Integrals Into Petri Nets, and Their High-Performance Execution Using GPI-Space

The computation of Feynman integrals is a fundamental challenge in high-energy physics, particularly for multi-loop calculations in quantum field theory. Efficient computational methods are necessary to handle the increasing complexity of these integrals.

This chapter presents a framework for automating the reduction of Feynman integrals using Integration-by-Parts (IBP) identities and sector decomposition. We introduce a novel approach that transforms the sector structure of a given Feynman integral into a directed acyclic graph (DAG) and subsequently generates a Petri net representation. The Petri net is then encoded in XPNet format, allowing its execution within the GPI-Space parallel workflow system. This enables efficient large-scale IBP reduction computations by distributing tasks across multiple computing nodes. Our implementation is built upon the `feynman.lib` [44] library in Singular and provides a scalable solution to IBP reductions in multi-loop Feynman integrals.

6.1 Introduction

Feynman integrals play a central role in perturbative quantum field theory, as they describe scattering amplitudes in high-energy physics. The evaluation of these integrals becomes increasingly complex for multi-loop diagrams, making traditional computational approaches inefficient.

Integration-by-Parts (IBP) identities provide a systematic method for reducing complicated Feynman integrals to a minimal set of master integrals [45, 46]. However, solving the large linear systems generated by IBP reductions requires significant computational resources.

To optimize IBP reduction, we introduce the concept of web sector structures, which categorize integrals into hierarchical structures based on their propagators. These sector structures naturally form a directed acyclic graph (DAG), which we utilize to construct a Petri net representation. The Petri net formalism allows us to define dependencies between integrals and execute reductions in parallel.

Our workflow consists of the following steps:

1. Constructing the sector structure of a Feynman integral as a DAG.
2. Generating a Petri net representation from the DAG.
3. Encoding the Petri net in XPNet format.
4. Executing the XPNet file using GPI-Space to perform parallel IBP reductions.

This approach enables high-performance IBP computations by the use of distributed computing and formal workflow management.

6.2 Petri Net Representation

A **Petri net** is a bipartite graph consisting of places (circles) and transitions (rectangles), where places represent resources (tokens), and transitions represent operations.

6.3 Definitions

6.3.1 Feynman Graph

A **Feynman graph** is a connected, oriented graph that represents the interaction of particles in perturbative quantum field theory. It consists of vertices, which correspond to interaction points, and edges, which represent propagators of virtual particles.

Example 6.3.1. *Example of Feynman graph.*

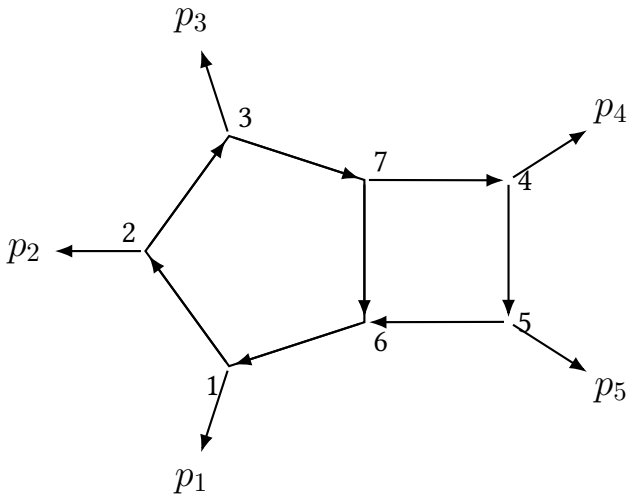


FIGURE 6.1: A two-loop five-point pentabox diagram with directed edges according to e.

6.3.2 Feynman Integral

(see

Given a Feynman graph, the associated Feynman integral takes the form:

$$G_{\alpha_1,\ldots,\alpha_n} = \int \prod_{j=1}^{L} \frac{\mathrm{d}^D l_j}{i\pi^{D/2}} \frac{1}{D_1^{\alpha_1} \cdots D_n^{\alpha_n}}, \tag{6.1}$$

where D_i are propagators corresponding to the edges of the graph, and α_i are integer indices [25] for more details).

6.3.3 Master Integral

A set of **master integrals** is a basis of integrals that are linearly independent under IBP identities, such that any Feynman integral in the family can be expressed as a linear combination of these master integrals. By construction, master integrals cannot be further reduced by IBP relations.

6.3.4 Target Integral

A **target integral** is a specific Feynman integral that we aim to reduce using IBP identities. The target integrals are chosen based on the physical problem being studied.

6.3.5 IBP Reduction

IBP reduction refers to the process of expressing target integrals as a linear combination of master integrals. This is achieved using IBP identities, which originate from the vanishing of total derivatives in dimensional regularization:

$$0 = \int \prod_{j=1}^{L} \frac{d^D l_j}{i\pi^{D/2}} \frac{\partial}{\partial l_k^\mu} \left(v^\mu \frac{1}{D_1^{\alpha_1} \cdots D_n^{\alpha_n}} \right), \tag{6.2}$$

where v^μ is a vector formed from loop or external momenta.

6.3.6 IBP Relations from the Baikov Representation

The Baikov representation rewrites Feynman integrals in terms of a new set of variables z_i:

$$G_{\alpha_1,\ldots,\alpha_n} = C \int \mathrm{d}z_1 \cdots \mathrm{d}z_n \, P^\gamma \frac{1}{z_1^{\alpha_1} \cdots z_n^{\alpha_n}}, \tag{6.3}$$

where P is the Gram determinant of the loop and external momenta. IBP relations follow from:

$$0 = C \int \mathrm{d}z_1 \cdots \mathrm{d}z_n \sum_{i=1}^{n} \frac{\partial}{\partial z_i} \left(a_i(z) P^\gamma \frac{1}{z_1^{\alpha_1} \cdots z_n^{\alpha_n}} \right). \tag{6.4}$$

The functions $a_i(z)$ satisfy syzygy conditions, enabling efficient IBP reduction.

6.3.7 Web Sector Structure

A web sector is defined by the presence or absence of propagators in a Feynman integral. It is represented as a binary vector:

$$s_i = \begin{cases} 1, & \alpha_i > 0 \\ 0, & \alpha_i \leq 0 \end{cases}. \tag{6.5}$$

Each sector is connected to its subsectors, forming a hierarchical directed acyclic graph (DAG) called the sector web. This structure enables efficient parallel computation of IBP reductions.

When the target integrals are given, their corresponding sectors are computed. The first step is to identify the integrals with the most positive indices, belonging to the highest sector. By iteratively changing indices from 1 to 0, all subsectors are generated, forming the sector web.

Each sector has two inputs:

1. The target integrals corresponding to the sector.
2. The tail integrals inherited from its supersectors.

Each sector undergoes the following steps:

1. Perform module intersection to generate IBP identities. Using the seed integrals from input and supersectors, substitute variables into the IBP system.
2. Truncate the IBP system and identify master integrals and tail integrals.
3. Store the reduced IBP relations and master integrals and propagate tail integrals to subsectors.

Example 6.3.2. *We have here an example of a web structure.*

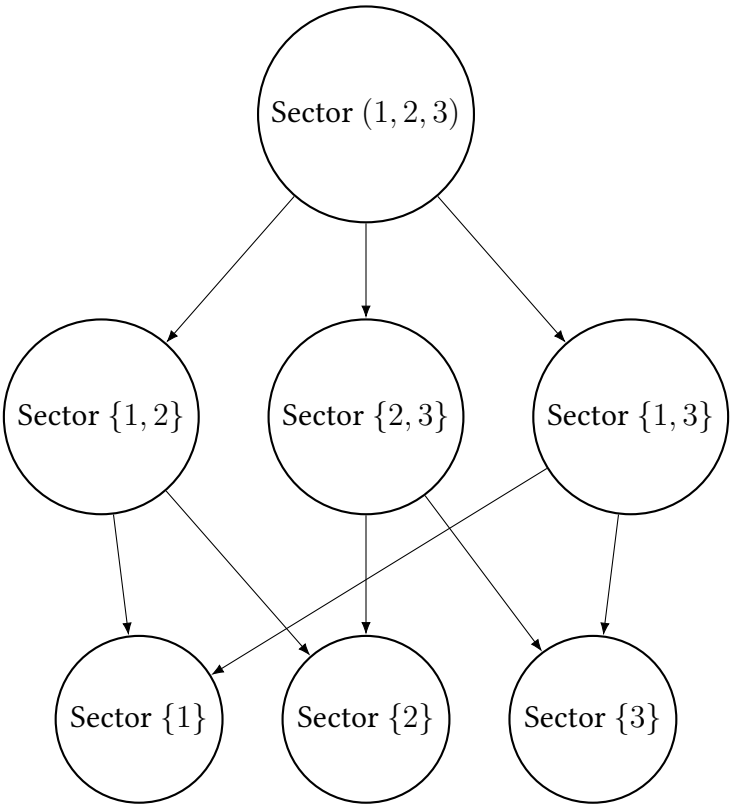


FIGURE 6.2: An example of a web structure of the sectors

6.4 IBP Computation Workflow

Given a Feynman graph and a set of target integrals, IBP reduction proceeds as follows:

1. Construct the Baikov representation of the Feynman integral.
2. Generate IBP relations by solving the syzygy conditions.
3. Organize integrals into a web sector structure by identifying the highest sector and constructing the sector web.
4. Apply IBP relations iteratively within each sector to reduce target integrals to master integrals.
5. Solve the resulting system of equations to express target integrals in terms of master integrals.

This workflow can be automated using symbolic algebra systems and parallel computing frameworks, significantly improving efficiency for multi-loop computations.

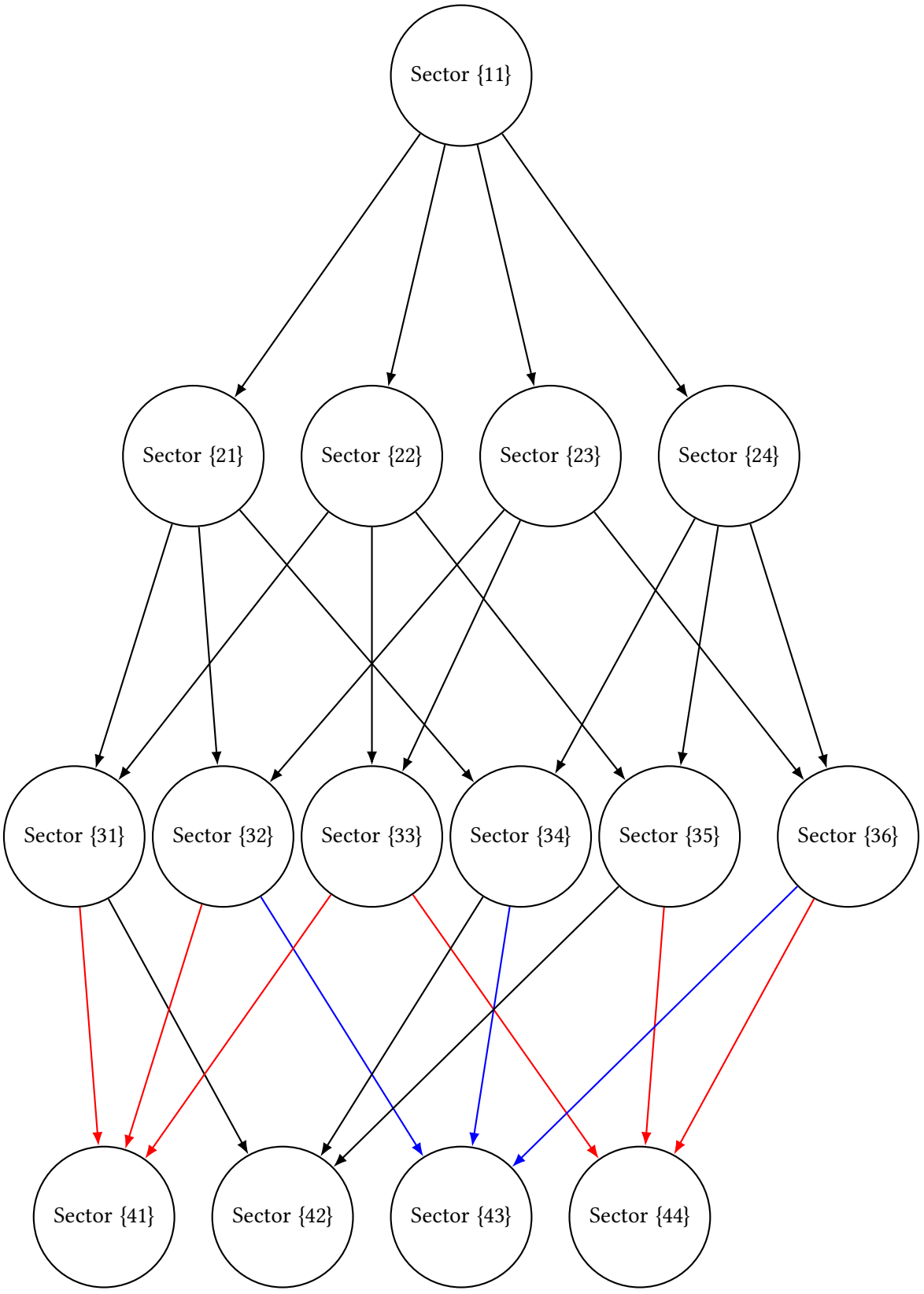


FIGURE 6.3: Web structure of sectors, illustrating hierarchical dependencies between computational nodes. Edges to nodes 41, 42, and 44 are colored red, and edges to node 43 are colored blue to highlight final connections.

6.5 Algorithms for Petri Net Generation

6.5.1 Generating a Directed Acyclic Graph (DAG)

Given a sector structure, we generate a Directed Acyclic Graph. We add T in front of the name since the transition's name should start with a letter in GPI-Space.

Algorithm 19: Generate Directed Acyclic Graph (DAG)

Input: Set of vertices V, Set of edges E

Output: Directed Acyclic Graph DAG

```
Initialize empty DAG dag;
foreach vertex v ∈ V do
    Append T_v to dag.vertices;
foreach edge (u, v) ∈ E do
    Append (T_u, T_v) to dag.edges;
return dag;
```

Algorithm 20: Generate Petri Net from DAG

Input: Directed Acyclic Graph DAG

Output: Petri Net Net

Initialize Petri Net Net;
Set $Net.transitions \leftarrow DAG.vertices$;
Initialize transition inputs T_{in} and outputs T_{out};
Initialize root nodes R as all vertices in DAG;
$R \leftarrow$ root node of DAG;
$T_{in}[r] \leftarrow$ places P_0 and $input$;
foreach *transition* $t \in DAG.vertices$ **do**
    Create control place $control_t$;
    $Net \leftarrow control_t$;
    $T_{in}[t] \leftarrow control_t$;
foreach *transition* $t \in DAG.vertices$ **do**
    Create place P_t for transition output;
    $T_{out}[t] \leftarrow P_t$;
foreach *edge* $(u, v) \in DAG.edges$ **do**
    $T_{in}[v] \leftarrow T_{out}[u]$
$Net \leftarrow T_{in}$;
$Net \leftarrow T_{out}$;
return Net;

Generating XPNet XML Representation

We first write the sector structure as a set of vertices and edges, where vertices are the sectors and edges are the links.

Algorithm 21: Generate XPNet XML from Petri Net

Input: Petri Net Net
Output: XML Representation of Petri Net

```
Initialize XML xml;
Append XML header and function definition to xml;
Identify last output place P_last from Net.transition_outputs;
Append input and output declarations to xml;
foreach place p ∈ Net.places do
    Append place declaration to xml;
foreach transition t ∈ Net.transitions do
    Append transition definition to xml;
    Append input connections to xml;
    Append output connections to xml;
    Append function call for t in XML format;
Append closing XML tags;
return xml;
```

Table 6.1: Example of Vertices and Edges

Vertices	Edges
$\{123,\ 12,\ 23,\ 13,\ 1,\ 2,\ 3\}$	$\{123, 12\}$
	$\{123, 23\}$
	$\{123, 13\}$
	$\{12, 1\}$
	$\{12, 2\}$
	$\{23, 2\}$
	$\{23, 3\}$
	$\{13, 1\}$
	$\{13, 3\}$

Table 6.2: Petri Net Data

Category	Details
Root Node	T_123

Category	Details
Transition Inputs	T_0 ← {control_T_0, library_name, base_filename, input} T_123 ← {control_123, input, web, tail_00} T_12 ← {control_12, input, web, tail_00, tail_123} T_23 ← {control_23, input, web, tail_00, tail_123} T_13 ← {control_13, input, web, tail_00, tail_123} T_1 ← {control_1, input, web, tail_00, tail_12, tail_13} T_2 ← {control_2, input, web, tail_00, tail_12, tail_23} T_3 ← {control_3, input, web, tail_00, tail_23, tail_13} T_end ← {control_T_end, tail_1, tail_2, tail_3}
Transition Outputs	T_0 → {labeledgraph, web, tail_00} T_123 → {tail_123} T_12 → {tail_12} T_23 → {tail_23} T_13 → {tail_13} T_1 → {tail_1} T_2 → {tail_2} T_3 → {tail_3} T_end → {output}

6.6 Petri Net Structure for Parallel IBP Reduction

The Petri Net depicted in Figure 6.4 models a workflow for Integration by Parts (IBP) reduction of Feynman integrals, designed to use parallel computation for efficiency. The net consists of transitions (cloud-shaped nodes: Sector 0, Sector 123, Sector 12, Sector 23, Sector 13, Sector 1, Sector 2, Sector 3, Sector end), places (circular nodes: P_0, P_123, P_12, P_23, P_13, P_1, P_-2, P_3, output), and control places (light-filled circles: control_T_0, control_123, control_12, control_23, control_13, control_1, control_2, control_3, control_T_end). Transitions represent computational tasks (e.g., computing IBP identities), places hold data (tokens) such as integrals, and control places provide conditions to fire transitions.

6.6.1 Workflow Description

The workflow aligns with a computational process for IBP reduction, as implemented in a program that processes sectors hierarchically. The key steps are:

- **Input Initialization**: The user provides a Feynman graph and target integrals (e.g., `tail_00`), stored in P_0. Sector 0 (transition) uses these to compute a Baikov representation and generate a sector web (`web_g`), producing initial tail integrals in P_0.
- **Sector Processing**: Each sector transition performs computations via `OneSectorComputation`, taking:
  - **Tail integrals** from supersectors, stored in places.
  - **Sector labels**, obtained via `getlabels` from the sector web, stored in control places.
  - **Labeled graph**, derived from the Feynman graph, also store in a place.
- **Computation**: Each transition executes `OneSectorComputation`, which:
  - Filters target integrals for the current sector using `getSector`.
  - Computes IBP identities (`computeManyIBP`) and reduces them (`getRedIBPs`) using the Gauss-Jordan elimination method.
  - Outputs a list containing reduced IBP identities (`reducedIBPs`), master integrals (`MIs`), and tail integrals (`OuputtailInts`).
- **Output Propagation**: Outputs are stored in places:

  - `reducedIBPs` and `MIs` are saved in the sector's output place for later use.
  - `OuputtailInts` are propagated to subsector transitions.

- **Final Output**: Sector end aggregates tail integrals from Sector 1, Sector 2, and Sector 3 (via P_1, P_2, P_3) to produce the final result in the `output` place.

Sector end collects results into the `output` place.

6.6.2 Necessity for Parallel Computation

The Petri Net's structure enables parallel computation, critical for scaling IBP reduction to large Feynman graphs:

- **Concurrent Transitions**: Transitions (e.g., Sector 12, Sector 23, Sector 13) can fire simultaneously once their input places (P_123) and control places (control_12, control_23, control_13) hold tokens.
- **Token-Driven Workflow**: Places hold tail integrals, triggering subsector transitions asynchronously when tokens are available, allowing concurrent execution of `OneSectorComputation` for sectors "1", "2", "3".
- **Dependency Management**: The net's edges reflect the program's dependency graph (e.g., sectorGraph).

6.6.3 Future Work: Sub-Petri Nets for Enhanced Parallelism

To further optimize parallelism, each cloud-shaped transition will be decomposed into a **sub-Petri net** in future work. Each sub-Petri net will model the internal steps of `OneSectorComputation` (e.g., `computeManyIBP`, `getRedIBPs`, filtering target integrals) as a network of smaller transitions and places. This decomposition aligns with the code's structure, where `OneSectorComputation` involves iterative loops and computations that can be parallelized. This approach ensures efficient and scalable execution of IBP reduction workflows, exploiting the parallel capabilities of GPI-Space.

To implement parallelization, we rewrote parts of the original `Singular` code in C++ and in `FLINT`, particularly for the Gaussian elimination function `gaussred_fq_nmod`. For a

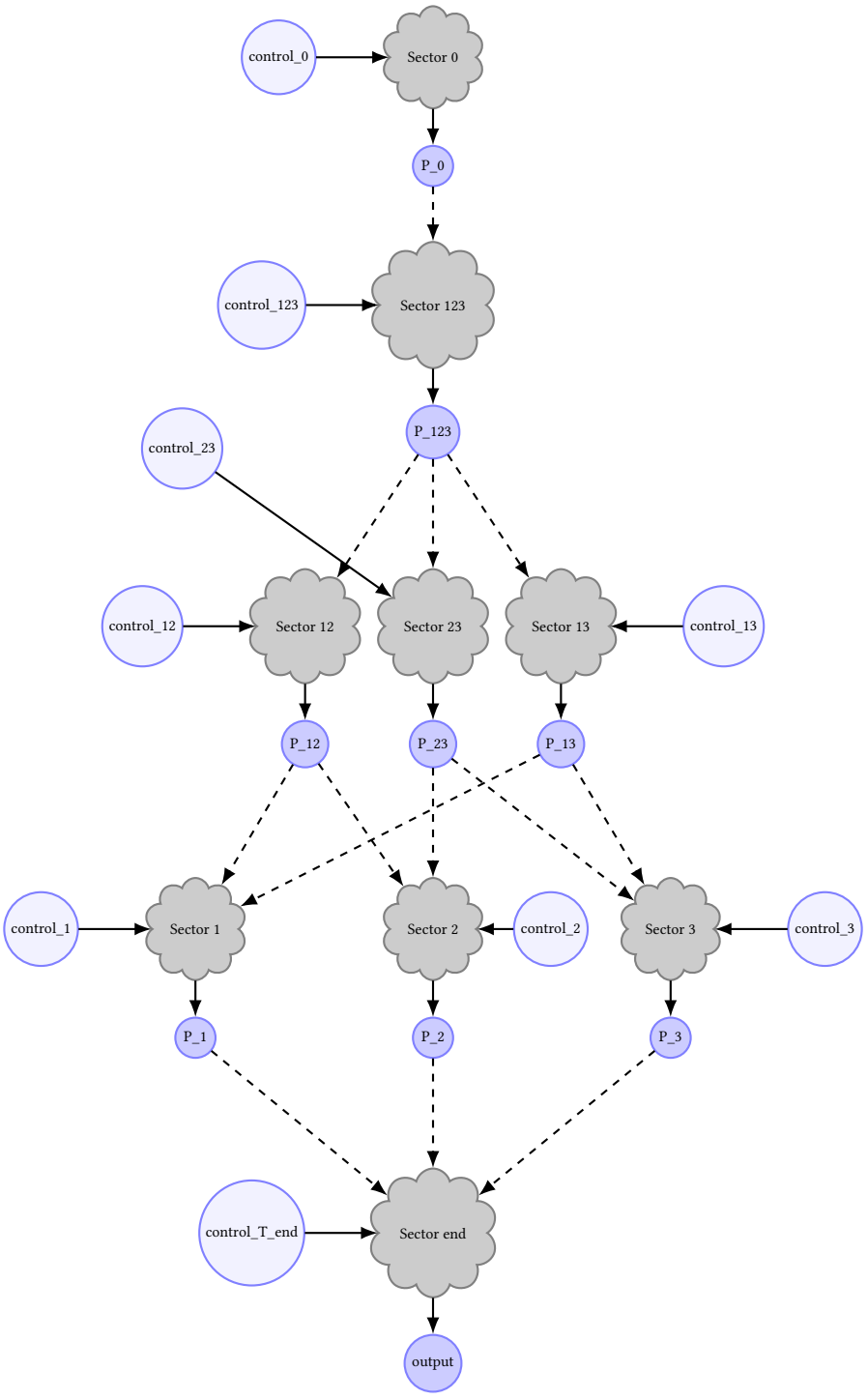


FIGURE 6.4: Petri Net Representation of web sector

given matrix A, this function computes the decomposition $P \times A = U \times S$, where P is a permutation matrix tracking row swaps, U is a unit lower triangular matrix storing elimination multipliers, S is the row echelon form, and the rank of A is determined. This allowed us to significantly improve computation times, as illustrated in Figure 6.6.

It can be observed that as we move deeper into the web structure, the computation time increases, since each child sector depends on information from its parent sectors to proceed. However, this hierarchical organization also provides flexibility: it enables the computation of a specific target sector without requiring the full structure to be processed. For instance, to compute sector {1}, we only need information from its immediate parents, sectors {1,2} and {1,3}.

Example 6.6.1. *Consider the graph*

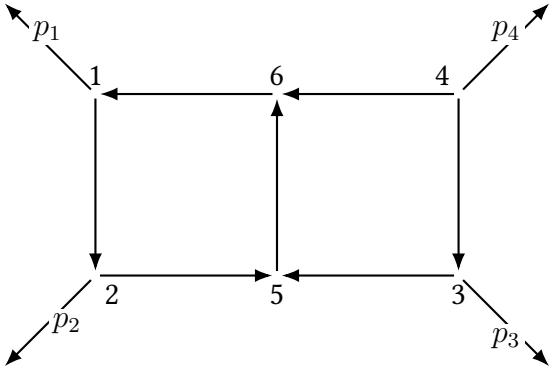


FIGURE 6.5: A two-loop Feynman graph with 6 vertices, 7 internal edges, and 4 external legs.

```
graph G =makeGraph(list(1,2,3,4,5,6),list(list(6,1),
list(4,6),list(1,2),list(3,5),list(4,3),list(2,5),list(5,6),
list(1),list(2),list(3),list(4)));

list tail_00 = list(list(1, 1, 1, -1, -3, -1, -1, -1, -1),
list(1, -1, 1, -1, -3, -1, -1, -4, -1));
```

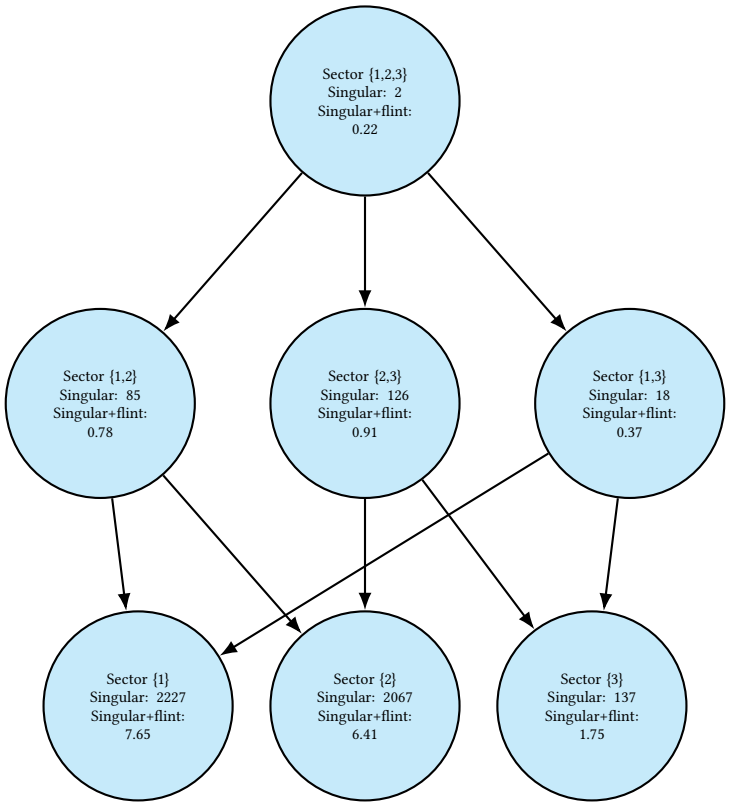


FIGURE 6.6: Web structure with timings for Sectors using Singular in sequential mode and Singular + flint in parallel mode.

In the following example, the target integral involves four non-negative indices. Due to memory limitations on our computer (16GB RAM), the computation for the individual sectors $\{1\}$, $\{2\}$, *and* $\{3\}$ *could not be completed.*

```
graph G = makeGraph(
    list(1,2,3,4,5,6),
    list(
        list(6,1), list(4,6), list(1,2),
        list(3,5), list(4,3), list(2,5), list(5,6),
        list(1), list(2), list(3), list(4)
    )
);

list tail_00 = list(
    list(1, 1, 1, 1, -3, -1, -1, -1, -1),
    list(1, -1, 1, -1, -3, -1, -1, -4, -1)
);
```

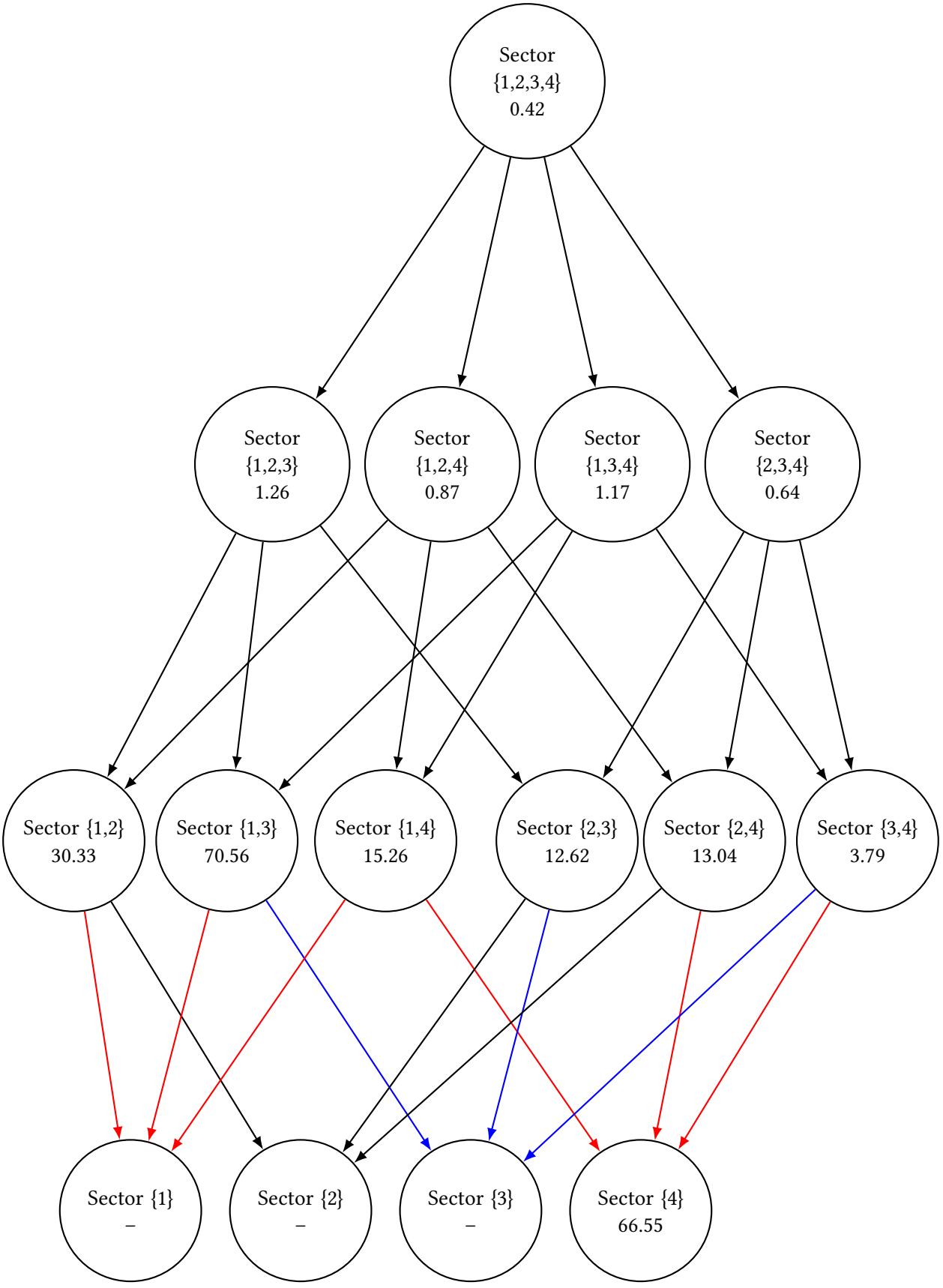


FIGURE 6.7: Web of sectors with computation times (in seconds) using Flint in GPI-Space. Colored edges highlight terminal sector dependencies.

Bibliography

[1] Diane Maclagan and Bernd Sturmfels. *Introduction to tropical geometry*, volume 161 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2015. [Cited on pages v, vi, and xiv.]

[2] Grigory Mikhalkin. Enumerative tropical geometry in $\mathbb{R}^2$. *J. Amer. Math. Soc.*, 18:313–377, 2005. [Cited on pages v, vi, vii, and 8.]

[3] B. Riemann. Theorie der Abel'schen Functionen. *Journal für die reine und angewandte Mathematik*, 54:115–155, 1857. [Cited on page v.]

[4] Hilbert. Über die Theorie der algebraischen Formen. *Mathematische Annalen*, 36:473–534, 1890. [Cited on page v.]

[5] Alexander Grothendieck. Éléments de géométrie algébrique (rédigés avec la collaboration de jean dieudonné) : II. étude globale élémentaire de quelques classes de morphismes. *Publications Mathématiques de l'IHéS*, 8:5–222, 1961. [Cited on pages v and vi.]

[6] Robin Hartshorne. *Algebraic Geometry*. Springer, 1977. [Cited on pages v and 9.]

[7] George M Bergman. The logarithmic limit-set of an algebraic variety. *Transactions of the American Mathematical Society*, 157:459–469, 1971. [Cited on page vi.]

[8] Oleg Viro. Dequantization of real algebraic geometry on logarithmic paper, 2000. [Cited on page vi.]

[9] William Fulton. *Introduction to Toric Varieties*. Number 131 in Annals of Mathematics Studies. Princeton University Press, 1993. [Cited on page vi.]

[10] Mark Gross. *Tropical Geometry and Mirror Symmetry*. Number 114 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, 2011. [Cited on page vi.]

[11] Philip Candelas, Xenia C. de la Ossa, Paul S. Green, and Linda Parkes. A pair of calabi–yau manifolds as an exactly soluble superconformal theory. *Nuclear Physics B*, 359(1):21–74, 1991. [Cited on page vi.]

[12] Maxim Kontsevich. Homological algebra of mirror symmetry. In *Proceedings of the International Congress of Mathematicians*, pages 120–139, 1995. [Cited on page vi.]

[13] Edward Witten. Phases of $N = 2$ theories in two dimensions. *Nuclear Physics B*, 403(1-2):159–222, 1993. [Cited on page vi.]

[14] Sergio Ferrara, Jeffrey A. Harvey, Andrew Strominger, and Cumrun Vafa. Second-quantized mirror symmetry. *Physics Letters B*, 361(1-4):59–65, 1995. [Cited on page vi.]

[15] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 2009. [Cited on page vii.]

[16] Robbert Dijkgraaf. *The moduli space of curves*, volume 129 of *Progr. Math.*, chapter Mirror symmetry and elliptic curves, pages 149–163. Birkhäuser Boston, 1995. [Cited on pages vii, 7, 13, 31, 48, and 49.]

[17] Janko Böhm, Kathrin Bringmann, Arne Buchholz, and Hannah Markwig. Tropical mirror symmetry for elliptic curves. *J. Reine Angew. Math.*, 732:211–246, 2017. [Cited on pages vii, xviii, 7, 8, 10, 11, 13, 20, 31, 48, 49, 56, 59, 71, and 72.]

[18] Janko Böhm, Christoph Goldner, and Hannah Markwig. Tropical mirror symmetry in dimension one. *SIGMA*, 18:046, 2022. Preprint, arXiv:1809.10659. [Cited on pages vii, 7, 19, 31, 49, and 57.]

[19] Si Li. *Calabi–Yau geometry and higher genus mirror symmetry*. PhD thesis, Harvard University, 2011. [Cited on page ix.]

[20] Si Li. Bcov theory on the elliptic curve and higher genus mirror symmetry, 2011. [Cited on pages ix and 19.]

[21] Renzo Cavalieri, Paul Johnson, Hannah Markwig, and Dhruv Ranganathan. Counting curves on Hirzebruch surfaces: tropical geometry and the Fock space, 2020. [Cited on page ix.]

[22] Hannah Markwig and Johannes Rau. Tropical descendant Gromov-Witten invariants. *Mathematische Zeitschrift*, 284(1-2):1–25, 2016. [Cited on page ix.]

[23] Andrei Okounkov and Rahul Pandharipande. Gromov-Witten theory, Hurwitz theory, and completed cycles. *Ann. of Math.*, 163(2):517–560, 2006. [Cited on pages ix, 14, and 57.]

[24] Janko Böhm, Christoph Goldner, and Hannah Markwig. Counts of (tropical) curves in $E \times \mathbb{P}^1$ and Feynman integrals. *Ann. Inst. Henri Poincaré D*, 9(1):121–158, 2022. Preprint, arXiv:1812.04936. [Cited on pages x, 7, 31, 49, and 59.]

[25] Zihao Wu, Janko Böhm, Rourou Ma, Hefeng Xu, and Yang Zhang. Neatibp 1.0, a package generating small-size integration-by-parts relations for Feynman integrals, 2024. [Cited on pages xi and 78.]

[26] Andreas Gathmann. *Gromov-Witten-and Degeneration Invariants: Computation and Enumerative Significance*. PhD thesis, Gottfried Wilhelm Leibniz Universität Hannover, Hannover, 1998. [Cited on pages xiv and 4.]

[27] Andreas Gathmann. Algebraic geometry. *Lecture Notes*, 2002. `https://agag-gathmann.math.rptu.de/class/alggeom-2002/alggeom-2002.pdf`. [Cited on pages xiv and 4.]

[28] Janko Böhm, Firoozeh Dastur, Alain Hoffmann, Hannah Markwig, and Ali Traore. Algorithms for Gromov-Witten invariants of elliptic curves. In *The Computer Algebra System OSCAR: Algorithms and Examples*, volume 32 of *Algorithms and Computation in Mathematics*, pages 167–190. Springer, 1 edition, 2024. [Cited on pages xiv and 31.]

[29] Ali Traore. Towards parallel algorithms for gromov-witten invariants of elliptic curves. In K. Buzzard, A. Dickenstein, B. Eick, A. Leykin, and Y. Ren, editors, *Mathematical Software – ICMS 2024*, pages 145–152. Springer Nature Switzerland, Cham, 2024. [Cited on pages xiv and 41.]

[30] W. Fulton and R. Pandharipande. Notes on stable maps and quantum cohomology, 1997. [Cited on page 5.]

[31] Kai Behrend and Barbara Fantechi. The intrinsic normal cone. *Invent. Math.*, 128:45–88, 1997. [Cited on pages 6 and 14.]

[32] Mark Gross and Bernd Siebert. Mirror Symmetry via logarithmic degeneration data I. *J. Differential Geom.*, 72:169–338, 2006. arXiv:math.AG/0309070. [Cited on pages 7 and 30.]

[33] Renzo Cavalieri and Eric Miles. *Riemann surfaces and algebraic curves*, volume 87 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 2016. A first course in Hurwitz theory. [Cited on page 8.]

[34] Renzo Cavalieri, Paul Johnson, and Hannah Markwig. Tropical Hurwitz numbers. *J. Algebr. Comb.*, 32(2):241–265, 2010. arXiv:0804.0579. [Cited on page 8.]

[35] Benoît Bertrand, Erwan Brugallé, and Grigory Mikhalkin. Tropical open Hurwitz numbers. *Rend. Semin. Mat. Univ. Padova*, 125:157–171, 2011. [Cited on page 11.]

[36] Kai Behrend. Gromov-Witten invariants in algebraic geometry. *Invent. Math.*, 127(3):601–617, 1997. [Cited on page 14.]

[37] Ravi Vakil. The moduli space of curves and Gromov–Witten theory. In *Enumerative invariants in algebraic geometry and string theory*, pages 143–198. Springer, 2008. [Cited on page 15.]

[38] Renzo Cavalieri, Paul Johnson, Hannah Markwig, and Dhruv Ranganathan. A graphical interface for the Gromov-Witten theory of curves. In *Algebraic geometry: Salt Lake City 2015*, volume 97 of *Proc. Sympos. Pure Math.*, pages 139–167. Amer. Math. Soc., Providence, RI, 2018. [Cited on page 17.]

[39] Janko Böhm, Kathrin Bringmann, Arne Buchholz, and Hannah Markwig. ellipticcovers.lib. A SINGULAR 4 library for Gromov-Witten invariants of elliptic curves, 2018. [Cited on page 20.]

[40] Firoozeh Dastur, Janko Böhm, Alain Hoffmann, Hannah Markwig, and Ali Traore. GromovWitten.jl: An OSCAR-based package computing Gromov-Witten invariants of elliptic curves, 2023. Available at `https://github.com/singular-gpispace/tropicalfeynman`. [Cited on page 24.]

[41] The FLINT team. *FLINT: Fast Library for Number Theory*, 2025. Version 3.2.1, `https://flintlib.org`. [Cited on page 41.]

[42] Elise Goujard and Martin Möller. Counting Feynman-like graphs: Quasimodularity and Siegel-Veech weight. arXiv:1609.01658, 2016. [Cited on pages 48, 49, and 50.]

[43] Georg Oberdieck and Aaron Pixton. Holomorphic anomaly equations and the Igusa cusp form conjecture. *Invent. Math.*, 213(2):507–587, 2018. [Cited on page 49.]

[44] Dushan Priyasad Honnaththara Acharige and Boehm Janko. Feynman IBP package, 2025. Version 0.1. [Cited on page 76.]

[45] K.G. Chetyrkin and F.V. Tkachov. Integration by parts: The algorithm to calculate β-functions in 4 loops. *Nuclear Physics B*, 192(1):159–204, 1981. [Cited on page 77.]

[46] S. Laporta. High precision calculation of multiloop Feynman integrals by difference equations. *Int. J. Mod. Phys. A*, 15:5087–5159, 2000. [Cited on page 77.]

Curriculum Vitae

Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau Kaiserslautern, Germany
Ph.D. in Mathematics 2021–Present

African Institute for Mathematical Sciences (AIMS) Cape Town, South Africa
Structured Master's Degree in Mathematics 2019–2020

- Thesis: *"Serre's Homological Criterion for Regularity of Noetherian Local Rings"*

Université Félix Houphouët-Boigny Abidjan, Côte d'Ivoire
Master in Mathematics 2017–2019

- Thesis: *"Hom-Lie Algebra Structures on Semi-Simple Lie Algebras"*

Faculté des Sciences et Techniques de Bamako Bamako, Mali
Licence en Mathématiques Appliquées 2010–2014

Wissenschaftlicher Werdegang

Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau Kaiserslautern, Deutschland
Promotion in Mathematik 2021–laufend

African Institute for Mathematical Sciences (AIMS) Kapstadt, Südafrika
Strukturiertes Masterstudium in Mathematik 2019–2020

- Masterarbeit: *"Serre's Homological Criterion for Regularity of Noetherian Local Rings"*

Université Félix Houphouët-Boigny Abidjan, Côte d'Ivoire
Master in Mathematik 2017–2019

- Masterarbeit: *"Hom-Lie-Algebra-Strukturen auf halbeinfachen Lie-Algebren"*

Faculté des Sciences et Techniques de Bamako Bamako, Mali
Bachelor in Angewandter Mathematik 2010–2014