Doctoral Thesis

Series Expansion Methods for Quantum Lattice Models

Author(s):
Hehn, Andreas R.

Publication Date:
2016

Permanent Link:
https://doi.org/10.3929/ethz-a-010742232

Rights / License:
In Copyright - Non-Commercial Use Permitted
SERIES EXPANSION METHODS FOR QUANTUM LATTICE MODELS

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by
ANDREAS ROLAND HEHN
Dipl.-Phys., Universität Stuttgart
born on 19.02.1985
citizen of
Germany

accepted on the recommendation of
Prof. Dr. Matthias Troyer, examiner
Prof. Dr. Simon Trebst, co-examiner

2016
Andreas Hehn

*Series Expansion Methods for Quantum Lattice Models*

Dissertation ETH No. 23724

DOI: 10.3929/ethz-a-010742232

E-mail: hehn@phys.ethz.ch
# Contents

Abstract vii  

Zusammenfassung ix  

## 1 Introduction 1  
1.1 Quantum Lattice Models .................................... 2  
1.2 Numerical Methods ............................................. 3  
1.3 Outline .......................................................... 5  

## 2 Series Expansions 7  
2.1 High-Temperature Series Expansions ......................... 8  
2.2 Perturbative $T=0$ Series Expansions ...................... 10  
2.2.1 Series Expansion for the ground state .................. 11  
2.2.2 Series Expansions for excited states ................... 12  
2.3 Linked Cluster Method ........................................ 15  
2.4 Series Analysis .................................................. 19  
2.4.1 Padé Approximation ......................................... 20  
2.4.2 DLog-Padé Approximation .................................. 21  
2.4.3 Argument Transformations .................................. 22  
2.4.4 Multivariate Approximation ................................ 24  

## 3 Graph Isomorphisms and Embeddings 25  
3.1 Graph Isomorphisms ............................................. 26  
3.1.1 McKay Algorithm .............................................. 27  
3.1.2 Colored Graphs ............................................... 33  
3.2 Graph Generation ............................................... 34  
3.3 Summary ......................................................... 36  

## 4 VLI — A Library for Large Integers and Polynomials 37  
4.1 Introduction ..................................................... 37  
4.2 Large Integers .................................................. 38
## CONTENTS

4.3 Polynomials ................................................................. 39
4.4 Optimized Inner Product ............................................... 39
  4.4.1 CPU Implementation .............................................. 40
  4.4.2 GPU Implementation .............................................. 40
4.5 Benchmarks ................................................................. 42
  4.5.1 Simple Integer Operations ..................................... 42
  4.5.2 Optimized Inner Product ...................................... 44
  4.5.3 Efficiency of the Optimized Inner Product ................. 47
4.6 Summary and Outlook .................................................. 48

5 High-T Series for the Heisenberg Model 49
  5.1 Introduction ............................................................ 49
  5.2 Model ................................................................. 50
  5.3 Running the Application ........................................... 51
    5.3.1 Graph Generation ............................................. 52
    5.3.2 Weight Computation .......................................... 53
    5.3.3 Subcluster-Subtraction and Embedding .................... 54
  5.4 Series Analysis ..................................................... 55
    5.4.1 Extraction of Coupling Constants ......................... 55
  5.5 Results ............................................................. 59
    5.5.1 Comparison to Quantum Monte Carlo ..................... 59
    5.5.2 Comparison with Cs$_2$CuBr$_4$ ............................ 60
  5.6 Extensibility ....................................................... 64
  5.7 Conclusion ........................................................... 65

6 T=0 Series for the Bose-Hubbard Model 67
  6.1 Introduction .......................................................... 67
  6.2 Bose-Hubbard Model ................................................ 69
  6.3 Running the Application ......................................... 72
    6.3.1 Graph Generation and Filtering ............................ 73
    6.3.2 Weight Computation .......................................... 74
    6.3.3 Subcluster-Subtraction and Embedding .................... 76
  6.4 Series Analysis ..................................................... 77
    6.4.1 Dispersion Functions ........................................ 78
    6.4.2 Gap Series and Critical Properties ...................... 79
  6.5 Results ............................................................. 81
    6.5.1 2D Lattices .................................................... 81
    6.5.2 Stacked Lattices .............................................. 82
  6.6 Extensibility ........................................................ 87
  6.7 Conclusion and Outlook .......................................... 89
CONTENTS

7 Conclusion ........................................... 91
A Computer Programs ............................... 95
B Series data ........................................ 97
  B.1 High-Temperature Series of the Heisenberg Model ................. 97
  B.2 Perturbation Series at $T = 0$ of the Bose-Hubbard Model ....... 99

Acknowledgments ..................................... 119

v
Abstract

In this thesis we study strongly correlated quantum many-body systems on regular lattices by means of high-order series expansions. We developed a C++ library which allows easy implementation of computer programs calculating high-temperature series and perturbative series at $T = 0$ for quantum magnets, bosonic and fermionic lattice models.

We present a high-temperature series expansion code for spin-1/2 Heisenberg models on arbitrary lattices. As an example we demonstrate how to use the computer programs for an anisotropic triangular lattice with two independent couplings $J_1$ and $J_2$ and compute the high-temperature series of the magnetic susceptibility and the static structure factor up to 12th and 10th order, respectively. We show how to extract effective coupling constants for the triangular Heisenberg model from experimental data on Cs$_2$CuBr$_4$.

For the Bose-Hubbard model we implemented a perturbative $T = 0$ series expansion code, which enables us to give quantitative estimates for the phase boundaries of the Mott phases in the ground state of the system on arbitrary lattices. Here we consider the triangular lattice, the square lattice and the honeycomb lattice, as well as, weakly coupled stacked layers of those lattice geometries. We calculate the 12th order ground state energy series of the $n = 1$ Mott state about the atomic limit. Furthermore, we calculate effective Hamiltonians for a single particle excitation and a single hole excitation from the Mott state up to 12th order. The same series expanded quantities are obtained for the $n = 2$ Mott state up to 8th order.

As these series expansion techniques rely heavily on high-precision integer and polynomial arithmetic, we also present a high-performance computing (HPC) library which performs the required operations efficiently on today’s supercomputers.
Zusammenfassung


Da die Reihenentwicklungsmethoden hauptsächlich auf Arithmetik mit Integern von hoher Genauigkeit und Polynomen beruht, präsentieren wir auch eine Programm bibliothek für zur effizienten Hochleistungsrechnung auf modernen Supercomputern.
Chapter 1

Introduction

One of the core objectives of condensed matter physics is to describe the physics that govern solid crystals. Even though the physics of single atoms are understood rather well, there are surprising phenomena when many of those atoms are brought together. They not only may form a regular crystalline structure, but also lead to emergent collective phenomena, which can not be inferred from the physics of the individual elements [5]. Two prominent examples of such emergent macroscopic phenomena are ferromagnetism or superconductivity.

However, due to the immense complexity of a macroscopic crystal containing $\sim 10^{23}$ atoms, each contributing several electrons, the system has to be radically simplified to attempt any microscopic comprehension of the macroscopic effects. Already in a classical theory the sheer number of particles renders a full direct treatment of the problem impossible. A proper quantum mechanical description of the system is beyond imagination. While the configuration space of the classical system grows linearly with the number of particles, the Hilbert space to describe a quantum mechanical system grows exponentially and yet alone storing the wavefunction $\psi$ on classical memory would require more atoms than the universe has to offer. Therefore, only extreme approximations allow us the slightest grasp of the world of condensed matter systems.

While complete comprehension of the effects in solids seems a hopeless challenge, physicists have been very successful in extracting the essential principles relevant for the macroscopic effect they were interested in. Often collective effects can be understood in a quasi-particle picture, where the macroscopic effect is described by emergent quasi-particles and their dynamics. Examples of such particles are phonons that represent lattice vibrations of the crystal, magnons which capture magnetic excitations in spin systems or Cooper pairs in the BCS theory of superconductors [6, 7]. Intriguingly, the emergent particles may have an entirely different nature than the individual physical particles of the underlying collective. In general, many approximations, even though they seem quite crude, yield sur-
prisngly good results and even allow quantitative predictions for materials and experiments.

The most renown quantitative method is Density Functional Theory (DFT) for which the late Walter Kohn received the Nobel Prize in Chemistry 1998 [8, 9, 10]. In a nutshell, the simplest form with the local density approximation (DFT+LDA) treats the electrons as single free particles in an effective potential created by the nuclei of the atoms and all other electrons. Iteratively minimizing the energy for each electron in the background created by all other electrons one finds a self-consistent solution for the orbital of each electron. Even though the approximation is uncontrolled, DFT+LDA has been very successful in chemistry and material science, where it is still widely used to predict crystal structures and electronic band structures [11].

Since the method does not capture the full physics of the interacting electrons, there are cases where it fails miserably. If the electrons are strongly correlated and can no longer be factorized into single electron descriptions, DFT may produce qualitatively wrong results. For example, according to the band structure predicted by density functional theory, NiO and many other transition-metal oxides are expected to exhibit a metallic ground state. However, due to the strong correlations of the electrons in the d and f shells the ground state is insulating and the electrons are localized even though the predicted bands are only partially filled. This is the so-called Mott insulating state [12, 13, 14], which can only be explained by models taking the strong electron-electron correlations into account. To understand these strong correlation effects and to explore the large variety of fascinating phenomena they offer, new approximations focusing on the strong interactions lead to quantum lattice models.

1.1 Quantum Lattice Models

Quantum lattice models are highly abstracted mathematical models of solid crystals where the most relevant atoms or even a whole unit cell is reduced to a fixed single point-like site with only a few local degrees of freedom repeated on an infinite regular lattice. In most cases the model describes the highest valence electron orbitals and the interaction of the electrons occupying these orbitals. While this model is a drastic simplification of the original system, it still poses a challenging problem and, hopefully, captures the essential physics to explain the phenomenon of interest.

A quantum lattice model consists of an infinite lattice of sites $i$, each spanning a local Hilbert space $\mathcal{H}_i$. The states of this Hilbert space may describe, for example, particles occupying the site or a quantum mechanical spin. The interactions of the
sites $i$ are governed by a Hamiltonian

\[ H = \sum_i H_i + \sum_{i,j} H_{i,j} + \sum_{i,j,k} H_{i,j,k} + \ldots, \]  

composed of single site terms $H_i$ and two-body interactions $H_{i,j}$, three-body interactions $H_{i,j,k}$ and so on. Since the considered electrons are highly localized at their respective atoms, and the dominating terms of the Hamiltonian are usually related to the hopping of these electrons with an hopping amplitude proportional to the overlap of the electron wave functions, often — and so we do in this thesis — only single site terms and nearest neighbor two-body interactions are considered. Two famous examples of quantum lattice models are the Heisenberg model, a successful toy model to describe magnetism of many materials, and the Hubbard model, describing positive charges in the superconducting CuO$_2$ planes of cuprates. For the Heisenberg model, each site of the lattice model represents a spatially fixed magnetic spin interacting with its neighboring spins on the lattice through an exchange interaction. The Hubbard model is closely related as it lead to an effective Heisenberg model in the strong-coupling limit. Here, the sites are vacancies which can be occupied by a single spin-$1/2$ particle, or two spin-$1/2$ particles of opposite direction. The particles can hop from sites to neighboring sites and are subject to a local repulsive potential.

### 1.2 Numerical Methods

With the increasing computational resources available to condensed matter physicists a great variety of numerical methods targeted at quantum lattice models have been developed within the last 50 years. A complete list is beyond the scope of this thesis and we limit the discussion to a subjective overview of a few common methods. None of the methods is able to offer a comprehensive picture of the physics in all lattice models, but each approach has its unique advantages and limitations. Thus, a system is usually studied with various techniques, each contributing a different perspective.

Conceptually, the most straightforward numerical approach to tackle these models is to diagonalize the Hamiltonian matrix. Unfortunately this method is restricted to very small system sizes, as the Hilbert space of the quantum system grows exponentially with the number of lattice sites and the Hamiltonian matrix quickly exhausts the available memory. Even “exact diagonalization” (ED), a more sophisticated approach, where the matrix is not stored explicitly and only a few eigen-values and their corresponding eigen-states are computed using iterative eigen-solvers, are limited to small system sizes of roughly 30–42 sites depending on the model [15, 16]. Consequently, the method is plagued by strong finite-size
1.2 Numerical Methods

effects and the results have to be extrapolated to the infinite lattice using finite-size scaling. In some cases the small systems may be unable to capture the behavior of the infinite lattice properly.

For one-dimensional systems the leading technique is the density renormalization group (DMRG) [17, 18]. Based on a matrix product state (MPS) ansatz, it provides a controllable approximation to compute static and dynamic properties of the ground state and low-lying excited states by truncating the entanglement of the system. Among many extensions are real-time evolution [19, 20, 21] and finite temperature calculations [22, 23]. For two-dimensional models the system gets mapped to a one-dimensional chain with long range interactions and the computational effort grows exponentially with the width [24]. A generalization of the ideas underlying the DMRG method to higher dimensions are tensor networks [25, 26], such as projected entangled pair states (PEPS). Even though these methods are still mostly experimental and under intensive development, they already obtained competitive results for some models [27].

Another remarkably successful group are Quantum Monte Carlo (QMC) methods. As its classical counterpart, Quantum Monte Carlo algorithms sample the partition function stochastically. This approach is effective for any number of spatial dimensions and allows simulations of very large system sizes. The sampling procedures depend on the particular method and range from sampling path integrals in real space with discrete (e.g. [28]) or continuous (imaginary) time (e.g. [29]) to diagrams in k-space [30]. One major drawback of Quantum Monte Carlo algorithms is the “sign problem” which makes a treatment of fermionic systems or frustrated magnets almost impossible [31]. The problem arises, for example, when two fermions change place which causes a sign-change and results in a negative weight of a configuration. It is worth mentioning, that there are some exceptions to this rule: a few special cases exists where fermionic systems can be examined without any sign-problem [32, 33].

In this thesis we discuss series expansion methods. As the name implies, this group of approaches expand a quantity, e.g. an observable, in power series of one or more parameters [34]. Series expansions have a long tradition in condensed matter physics [35] and have been applied to a large variety of systems from the classical Ising model [35], quantum magnets [36] to bosonic [37] and fermionic [38, 39] lattice models. The methods work well for arbitrary dimension and yield results in the thermodynamic limit. The main limitation is a finite truncation order of the expanded series restricting the range of the expansion parameters for which the series is a valid approximation.

We implement and apply two complementary series expansion approaches, the high-temperature series expansion for thermodynamic properties and the perturbative series expansion at \( T = 0 \) for ground state and low-lying excited state properties.
1.3 Outline

We start by introducing the theoretical foundations of high-temperature series expansions and perturbative series expansions at $T = 0$ in chapter 2. An essential role in both methods plays the linked cluster expansion: by splitting the calculation of the series for the infinite lattice into high-order series derivations on many small pieces of the lattice, the linked cluster expansions allows us to obtain results directly in the thermodynamic limit, without finite-size effects. The limitation of the series expansions methods is the finite expansion order, where a typical high-order calculation may yield a series of $10^{th}$–$20^{th}$ order, depending on the complexity of the model. Since in many cases the convergence radius of such a perturbation series is small and only valid close to the unperturbed limit, we also introduce some mathematical tools to analytically extrapolate the series beyond this limit and extract information of the underlying function, such as the critical behavior, from the few coefficients of the series we could obtain.

A cornerstone of the linked cluster method, which splits the lattice into small graphs, is the reduction of the computational effort by identifying isomorphic graphs and computing their contribution only once. In chapter 3 we describe an algorithm to identify isomorphic graphs, and explain how to generate the list of all contributing graphs from an infinite lattice.

The computation of the contribution of each graph to the high-temperature expansion or the perturbative $T = 0$ expansion, relies heavily on three data structures: polynomials, high-precision integers and vectors thereof. In chapter 4 we present a high-performance computing (HPC) library providing such data structures including highly optimized kernels for basic operations on those entities.

In the last two chapters we assemble all pieces and turn to applications. In chapter 5 we present a set of computer programs to calculate the high-temperature series expansion of the free energy, the magnetic susceptibility and the static magnetic structure factor for Heisenberg models on arbitrary lattices. As a showcase study we calculate the high-temperature series for the Heisenberg model on an anisotropic triangular lattice with two independent couplings. By fitting the obtained series for the magnetic susceptibility to measurements on Cs$_2$CuBr$_4$ we extract effective coupling constants for the model of the material.

In chapter 6 we map out the ground state phase diagram of the Bose-Hubbard model on various two- and three-dimensional lattices by means of perturbative series expansions at $T = 0$. Starting from the gapped Mott state in the atomic limit, we perturbatively switch on the interactions between the sites. From the series of the single-particle and single-hole excitation we extract a quantitative prediction for the boundaries of the Mott phase, study the critical behavior of the gap closing at commensurate density and compare our findings with results from Quantum Monte Carlo and other approaches.
1.3 Outline
Chapter 2

Series Expansions

Series expansions are a whole group of computational methods that calculate the coefficients $a_i$ of a power series of the functional behavior of a physical quantity $f(x)$ exactly up to an order $N$

$$f(x) \approx a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N. \quad (2.1)$$

The methods differ in the meaning of the parameter $x$, and consequently what quantities $f(x)$ are of interest, or in how the series are computed. In contrast to other computational approaches based on a power series expansion, such as the Stochastic Series Expansion (SSE) [28] or diagrammatic Monte-Carlo (DiagMC) [30], series expansion methods have the unique feature that the coefficients are calculated analytically instead of being estimated statistically. In fact, in most cases the coefficients are even computed using infinite precision arithemtics without any numerical error.

In this thesis we discuss two of those methods, namely high-temperature series expansions and perturbative series expansions at zero temperature. The former method expands thermodynamic quantities, such as the magnetic susceptibility, in the inverse temperature $\beta$ and the latter expands the ground-state energy or an effective Hamiltonian of low-lying excited states perturbatively in a small parameter $x$. A key role in the calculation procedure of both methods plays the linked cluster method, which allows to obtain series directly in the thermodynamic limit as explained in section 2.3. In general only the lowest orders of a series are accessible through series expansion methods limiting the radius of validity to a narrow region around $\beta = 0$ or $x = 0$. The final section of this chapter is dedicated to the question how to extract interesting properties of the underlying function $f(x)$ of a physical quantity from only a few coefficients of its series.
2.1 High-Temperature Series Expansions

High-temperature series expansions have a long tradition in condensed matter physics [35, 34]. As the name implies, the idea is to expand a thermodynamic quantity $Q$ in the inverse temperature $\beta = 1/(k_B T)$

$$Q(\beta) = \sum_n a_n \beta^n$$  \hspace{1cm} (2.2)

around the infinite temperature limit $\beta = 0$. At the heart of high-temperature series expansions lies the expansion of the partition function

$$Z = \text{Tr}_\mathcal{H} e^{-\beta H} ,$$  \hspace{1cm} (2.3)

where the trace runs over the full Hilbert space $\mathcal{H}$ of the system described by the Hamiltonian $H$. From the partition function $Z$ all thermodynamic quantities, including susceptibilities or correlators, are accessible. To evaluate this simple looking function two problems have to be overcome: first, a direct evaluation of the exponential of the operator $H$ requires the diagonalization of the operator, second, in the thermodynamic limit, where the lattice is of infinite size, the dimension of the Hilbert space $\mathcal{H}$ is of infinite dimension.

The high-temperature series is obtained by simply expanding the exponential function

$$Z \approx \sum_{n=0}^N \frac{(-\beta)^n}{n!} \text{Tr}_\mathcal{H} H^n ,$$  \hspace{1cm} (2.4)

where, again, the trace covers the full Hilbert space of the quantum system. While the series expansion obviously addresses the problem of evaluating the exponential function, the infinite dimension of the Hilbert space remains. Thus one might wonder how the series expansion makes the evaluation feasible. The key are the finite truncation order $N$ and, assuming $H$ is composed of local interactions only, a finite
interaction range of $H^n$-terms in combination with the translational invariance of the infinite lattice. These features, being the limitations of the method at the same time, allow to compute the series of the partition function for the infinite system exactly up to a given order $N$ by calculating and combining the contributions of a set of small pieces of the lattice (fig. 2.1) which capture the finite interaction range of the $H^n$-terms.

There are several methods how to construct the small lattice pieces — more commonly referred to as “clusters” or “graphs” — and combine their contributions to obtain the series for the full, infinite lattice. Here, we employ the linked-cluster expansion method which we explain in section 2.3. With the size of the largest graph to be considered being directly proportional to the desired order $N$, the number of contributing graphs is typically between a few thousand and several millions, depending on the lattice structure.

Computationally, the core of the method is the calculation of the partition function series for each contributing graph. While computing the trace is conceptually straight-forward, namely applying $n$ times the operator $H$ to each state $|\psi\rangle$ of the trace, the exponentially increasing dimension of the Hilbert space with the number of sites in the graph make the computations challenging and eventually limits the method to small orders $N$.

To compute susceptibilities the original Hamiltonian $H_0$ of the system is extended by an auxiliary field term

$$H = H_0 + hH_{\text{ext}},$$

where $H_{\text{ext}}$ is the operator controlled by the external field $h$. The series for the corresponding susceptibility at $h \to 0$

$$\chi = \beta^{-1} \frac{\partial^2}{\partial h^2} \ln Z$$

$$= \beta^{-1} \frac{Z_2}{Z_0} + O(h)$$

contains two terms: the series expanded partition function $Z_0$ of the original Hamiltonian $H_0$ and the modified partition function $Z_2$ which contains two occurrences of the auxiliary field operator

$$Z_2 = \sum_{n=0}^{N} \frac{(-\beta)^n}{n!} \frac{\text{Tr}}{n!} \sum_{m=0}^{n} H_0^{n-m} H_{\text{ext}} H_0^m H_{\text{ext}}.$$

Aside from the susceptibility, also structure factors are of general interest, as they can be directly measured by neutron scattering experiments and provide a
2.2 Perturbative T=0 Series Expansions

A valuable link between theory and experiments. The static structure factor can be obtained through Fourier transformation

\[ S(k) = \frac{1}{N} \sum_{i,j} \langle O_i O_j \rangle e^{-i k (x_j - x_i)} \]  \hspace{1cm} (2.9)

from the real space equal-time correlators \( \langle O_i O_j \rangle \) between sites \( i \) and \( j \). The correlators are obtained by series expanding the density operator \( \rho \) of the canonical ensemble, similar to the expansion of the partition function

\[ \langle O_i O_j \rangle = \text{Tr}_\mathcal{H} \rho O_i O_j \]  \hspace{1cm} (2.10)

\[ = \frac{1}{Z} \text{Tr}_\mathcal{H} O_i O_j e^{-\beta H} \]  \hspace{1cm} (2.11)

\[ = \frac{1}{Z} \sum_{n=0}^{\infty} \frac{(-\beta)^n}{n!} \text{Tr}_\mathcal{H} O_i O_j H^n. \]  \hspace{1cm} (2.12)

2.2 Perturbative T=0 Series Expansions

The underlying principle of perturbative series expansions at \( T = 0 \) is taught in every basic quantum mechanics class as Rayleigh-Schrödinger perturbation theory or just perturbation theory [40]. Starting from an exactly solvable system with Hamiltonian \( H_0 \), the system is perturbed by the “perturbation” operator \( V \) leading to a Hamiltonian of the form

\[ H = H_0 + x V. \]  \hspace{1cm} (2.13)

The small continuous parameter \( x \) drives the perturbation and serves as the expansion variable. The second order formula for the energy of a non-degenerate eigenstate \( |k\rangle \) reads

\[ E = E_k^0 + x \langle k | V | k \rangle + x^2 \sum_{m \neq k} \frac{| \langle m | V | k \rangle |^2}{E_k^0 - E_m^0}, \]  \hspace{1cm} (2.14)

where \( E_k^0 = \langle k | H_0 | k \rangle \) are the energies of the unperturbed system.

The perturbative series expansions at \( T = 0 \) take this method one step further and generate series for the ground-state energy and an effective Hamiltonian for low-lying excited states up to 10\(^{th}\) – 20\(^{th}\) order. Like for the high-temperature series we limit our discussion to perturbative series expansions of quantum lattice models. Furthermore, the perturbative series expansion method we discuss here, also includes dividing the computation of the series for the infinite lattice into
a set of several small graphs and combining their result using the linked cluster method discussed in the next section. In fact, in the literature the perturbation method itself is often referred to as “linked cluster expansion” (LCE) or “linked cluster method” as it is tightly coupled to the graph formalism with the same name. As we employ this graph formalism in the context of perturbative $T = 0$ series expansions as well as high-temperature series expansions, we use the term “linked cluster method” exclusively for the graph formalism and refer to the perturbation expansion as “perturbative $T = 0$ series expansions” or “ground-state series expansions”. A more detailed review on the method can be found in the references [34] and [41]. Even though the latter reference is a bit dated and misses the important extension to multi-particle excitations developed shortly after the review was published [43, 44], it still provides a good introduction to the method.

For the linked cluster approach we use here, the lattice should be chosen in such a way, that the exactly solvable $H_0$ operators describe the disconnected sites of the lattice and the perturbation $xV$ the bonds between those sites.

### 2.2.1 Series Expansion for the ground state

The derivation of the ground state series for higher orders follows the Rayleigh-Schrödinger perturbation theory. The calculation is performed in the eigen-basis of the unperturbed, solved system

$$H_0 |k\rangle = \varepsilon_k |k\rangle,$$

where the set of eigen-vectors $\{|k\rangle\}$ form a full orthonormal basis of the Hilbert space $\mathcal{H}$. While we allow degeneracies $\varepsilon_k = \varepsilon_{k'}$ in general, we assume a non-degenerate ground-state $|0\rangle$. For the lattice models considered here, $\{|k\rangle\}$ is conveniently chosen as the set of product-states of the single site basis states, and we require the unperturbed system to represent disconnected non-interacting lattice sites. The ground-state of the full unperturbed Hamiltonian is then the product state in which all sites are in their individual ground-state $|0\rangle = |0\rangle_1 \otimes \cdots \otimes |0\rangle_L$.

To solve the perturbed Hamiltonian $H$ (2.13), we make a power series ansatz for the ground-state energy

$$E = \sum_n E_n x^n$$

and the ground state

$$|\psi\rangle = \sum_n x^n |\psi_n\rangle$$

of the full Hamiltonian and choose the normalization

$$\langle 0 | \psi_n \rangle = \delta_{0,n}.$$
2.2 Perturbative $T=0$ Series Expansions

From the Schrödinger equation

$$(H_0 + xV)|\psi\rangle = E|\psi\rangle,$$  \hspace{1cm} (2.19)

we obtain a set of equations, by collecting terms with same powers of $x$. For each order $n$, these equations can be solved recursively as

$$E_n = \langle 0 | V | \psi_{n-1} \rangle,$$  \hspace{1cm} (2.20)

$$\langle k | \psi_n \rangle = \frac{1}{\varepsilon_0 - \varepsilon_k} \left( \langle k | V | \psi_{n-1} \rangle - \sum_{i=1}^{n} E_i \langle k | \psi_{n-l} \rangle \right),$$  \hspace{1cm} (2.21)

yielding the coefficients $E_n$ of the ground-state energy series (2.16). For the high-order series expansion the recursive evaluation of those two equations for a small finite graph forms the core of the algorithm. Beside the direct result of the expansion, the ground-state energy series $E$ of the perturbed system, it is also possible to compute the series of the ground-state expectation value of an arbitrary operator $O$ from the obtained series of states $|\psi\rangle$

$$\langle O \rangle = \frac{\langle \psi | O | \psi \rangle}{\langle \psi | \psi \rangle}.$$  \hspace{1cm} (2.22)

2.2.2 Series Expansions for excited states

The key idea to obtain properties of low-lying excited states is to perturbatively derive effective Hamiltonians for each quasi-particle sector [42, 43, 44]. As we use a perturbative approach, the quasi-particles originate from the low excitations of unperturbed single sites and get “dressed” by virtual excitations due to the perturbation. For example, the single-particle states of an unperturbed system of a graph with $L$ sites are the $L$ states in which one of the sites $l$ is in its first excited state, while all other sites remain in their ground-state $|0\rangle_l$.

Given the diagonal Hamiltonian $H_0$ of the unperturbed system and the arbitrarily structured perturbation operator $V$, the goal is to perturbatively generate
an effective Hamiltonian which is block-diagonal up to a given order $N$

$$
\begin{pmatrix}
\varepsilon_{(0p)} & 0 & \cdots & 0 \\
0 & \varepsilon_{(1p)} & \cdots & 0 \\
0 & 0 & \ddots & \ddots \\
0 & 0 & \ddots & \varepsilon_{(2p)}
\end{pmatrix} + \begin{pmatrix} xV \end{pmatrix}
$$

(2.23)

where each block $H_{\text{eff}}|_0$, $H_{\text{eff}}|_1$, $H_{\text{eff}}|_2$, \ldots conserves the number of particles and is restricted to the particle sector, containing 0, 1, 2, \ldots particles, respectively.

The perturbative block-diagonalization process is not unique. Most notably, the following three methods are used: the initial similarity-transformation approach by Gelfand [42], which was the first to compute the single-particle sector, and, following the terminology of [34], the more general two-block orthogonal transformation (TBOT) and multiblock orthogonal transformation (MBOT) methods by Trebst et al. [43, 44]. While the first two techniques block-diagonalize the matrix in only two blocks, one block being the effective Hamiltonian of the particle sector of interest, the other absorbing the remaining system, the last method fully block-diagonalizes the Hamiltonian of a finite graph up to a given order.

**Similarity Transformation Method**

The original method by Gelfand aims at obtaining the effective Hamiltonian of the single-particle sector by employing a general similarity transformation

$$
H_{\text{eff}} = S^{-1} HS,
$$

(2.24)

with only a single requirement: for a graph with $L$ sites, having $L$ first excited states (in a particular symmetry sector) of the unperturbed Hamiltonian $H_0$, the transformation $S$ should be the identity for the $L \times L$ one-particle block. This
ensures that the states within the manifold of first excited states do not mix and that the resulting effective Hamiltonian will decompose into an $L \times L$ block describing the physics of the one-particle sector and another block capturing the remaining physics on the graph. Again, the requirement on $S$ and the block decomposition will only be satisfied up to a certain order $N$.

Let us denote the $L$ degenerate unperturbed first excited states of this sector by $|l\rangle$, where $l \in [1, L]$. As we are only interested in the one-particle block of the effective Hamiltonian, we will only concern ourselves with the corresponding columns of $S = (|\psi^{(1)}\rangle, \ldots, |\psi^{(L)}\rangle)$. Multiplying (2.24) by $S$ from the left, we can construct a set of equations for the matrix elements of the $L \times L$ part of the effective Hamiltonian

$$
\sum_{l'} |\psi^{(l')}\rangle \langle l'| H_{\text{eff}} |l\rangle = H |\psi^{(l)}\rangle.
$$

Making series ansatzes for the columns of $S$ and the effective Hamiltonian

$$
|\psi^{(l)}\rangle = \sum_n x^n |\psi^{(l)}_n\rangle,
$$

$$
H_{\text{eff}} = \sum_n x^n H_{\text{eff}}^n,
$$

together with the Hamiltonian (2.13), we obtain a set of equations for the matrix elements of $H_{\text{eff}}$ in the basis of the unperturbed single-particle states $\{|l\rangle = |\psi^{(l)}_0\rangle\}$ by collecting same powers of $x$

$$
\sum_{n'=0}^{L} \sum_{l'=1}^{L} |\psi^{(l')}_{n'}\rangle \langle l'| H_{\text{eff}}^{n-n'} |l\rangle = H_0 |\psi^{(l)}_n\rangle + V |\psi^{(l)}_{n-1}\rangle.
$$

With the requirement on the transformation for the $L \times L$ single-particle block, which leads to a generalization of the normalization (2.18) we choose for the ground-state expansion

$$
\langle l|\psi^{(l')}_{n}\rangle = \delta_{0,n} \delta_{l,l'},
$$

we obtain a set of recursive relations for the matrix elements of the one-particle block of the effective Hamiltonian

$$
\langle l'| H_{\text{eff}}^n |l\rangle = \langle l'| V |\psi^{(l)}_{n-1}\rangle,
$$

$$
\langle k| \psi^{(l')}_{n}\rangle = \frac{1}{\varepsilon_{(1p)} - \varepsilon_k} \left( \langle k| V |\psi^{(l)}_{n-1}\rangle - \sum_{n'=1}^{n-1} \sum_{l'=1}^{L} \langle l'| H_{\text{eff}}^{n-n'} |l\rangle \langle k| \psi^{(l')}_{n'}\rangle \right),
$$

where $|k\rangle$ are the unperturbed states outside the one-particle sector and $\varepsilon_{(1p)}$ is the energy of the degenerate unperturbed first excited states $\varepsilon_{(1p)} = \varepsilon_l = \langle l| H_0 |l\rangle$. These equations are very similar to the equations of the ground-state expansion and can again be solved recursively to construct $H_{\text{eff}}$ order by order.
Multiblock Orthogonal Transformation Method

The multiblock orthogonal transformation [43, 44] generalizes the idea of the similarity transformation to obtain an effective Hamiltonian $H_{\text{eff}}$ by requiring the transformation to be unitary. This preserves all symmetries of the original Hamiltonian and overcomes the shortcomings of the simpler similarity approach, which fails if the ground state and the excited state sector have the same quantum numbers. For the case of a real Hamiltonian, the unitary transformation reduces to an orthogonal transformation

$$H_{\text{eff}} = O^T H O,$$  \hspace{1cm} (2.32)

where the orthogonality can be ensured by

$$O = e^S,$$  \hspace{1cm} (2.33)

if $S$ is an antisymmetric real matrix $S^T = -S$.

2.3 Linked Cluster Method

The linked-cluster expansion is based on the idea to express an extensive quantity $A$ on the lattice $L$ as a sum of contributions, so-called irreducible “weights” $w^{(g)}$, of small connected parts of the lattice, the embeddable connected graphs or clusters $g \in G_L$. As a graph $g$ appears several times in the lattice the weight $w^{(g)}$ has to be multiplied by the number of distinguishable embeddings in the lattice, the so-called lattice constant $\text{LC}(g)$

$$A = \sum_{g \in G_L} \text{LC}(g) \cdot w^{(g)}.$$  \hspace{1cm} (2.34)

For infinite, translational invariant lattices the quantity $A$ is usually expressed per site, in which case the embeddings are also counted per site.

The same idea applies to the graphs themselves. The quantity $A^{(g)}$ defined on a graph $g$ is again expressed as a sum of contributions of embeddable graphs $g'$

$$A^{(g)} = \sum_{g' \subseteq g} c(g', g) \cdot w^{(g')}.$$  \hspace{1cm} (2.35)

Like the lattice constant, $c(g', g)$ denotes in how many ways graph $g'$ can be embedded in graph $g$. Note that the sum includes the graph $g$ itself. Until here, this is merely an interesting theoretical construct on how to compose $A^{(g)}$ from graph contributions, but we do not know how to calculate the irreducible weights.

However, we may assume, for a small graph $g$, that computing quantity $A^{(g)}$ is directly possible. In the high-temperature series expansion, this may be the series
2.3 Linked Cluster Method

Figure 2.2: Pictorial representation of the subcluster-subtraction of a graph with five vertices. Graphs composed of solid sites represent the reducible quantity $A^{(g)}$, graphs with hollow circle sites represent irreducible weights $w^{(g')}$.  

for the free energy $F_g = \beta^{-1} \ln Z_g$ or the susceptibility $\chi_g$ computed from a trace over the finite Hilbert space $\mathcal{H}_g$. Starting from a graph $g_1$ of only a single vertex without any edges, where

$$A^{(g_1)} = w^{(g_1)}, \tag{2.36}$$

equation (2.35) defines a recursive scheme to determine the irreducible weights $w^{(g)}$ — the so-called subcluster-subtraction: given the extensive quantity $A^{(g)}$ of each graph $g$ we can obtain the irreducible weights $w^{(g)}$ by recursively subtracting the weights $w^{(g')}$ of all graphs $g'$ embeddable in $g$

$$w^{(g)} = A^{(g)} - \sum_{g' \subset g} c(g', g) \cdot w^{(g')} \tag{2.37}$$

An example subcluster-subtraction for a graph with five vertices is depicted in figure 2.2. Once the irreducible weights $w^{(g)}$ are known, they may be combined to $A$ defined on the full lattice (eq. 2.34).

For an infinite lattice the number of contributing graphs $g$ is evidently infinite. However, due to the finite range of the $H^n$-terms in a high-temperature series expansion all irreducible weights $w^{(g)}$ for graphs with more than $n$ edges hosting a pair-wise interaction, such as the $S_i S_j$ interaction of the Heisenberg model, vanish. Hence, for a finite expansion order $N$ we only need to consider a small fraction of all embeddable graphs $G_L^*$, namely all embeddable graphs with $n \leq N$ edges.

Furthermore, we can show that only embeddable graphs which are connected by at least one edge (bond) have to be considered: Given a disconnected graph $g = g_a \cup g_b$, where $g_a$ and $g_b$ do not share any edges (bonds) nor vertices (sites) $g_a \cap g_b = \emptyset$, including their irreducible weights and the irreducible weights of all
the graphs embeddable in \( g_a \) and \( g_b \), performing the subcluster-subtraction (2.37) for \( g \) yields

\[
w^{(g)} = A^{(g)} - w^{(g_a)} - w^{(g_b)} - \left( \sum_{g' \in G_a} c(g', g_a) \cdot w^{(g')} \right) - \left( \sum_{g' \in G_b} c(g', g_b) \cdot w^{(g')} \right),
\]

where \( G_a \) and \( G_b \) are the graphs embeddable in \( g_a \) and \( g_b \), respectively. Inserting the subcluster-subtraction leading to the weights of \( g_a \) and \( g_b \) we obtain

\[
w^{(g)} = A^{(g)} - A^{(g_a)} - A^{(g_b)} = 0,
\]

since we required \( A \) to be extensive. Therefore, only connected graphs \( G_L \subset G^*_L \) have a finite irreducible weight, which is why the method is called connected or linked cluster expansion [41].

**Linked Cluster Expansion for Operators**

For the cluster expansion of the perturbative series expansions at \( T = 0 \) we have to further investigate the effective Hamiltonian \( H_{\text{eff}} \). Following the detailed discussion by C. Knetter [45], we can decompose the effective Hamiltonian on the Hilbert space of the full lattice into \( n \)-particle irreducible parts

\[
H_{\text{eff}} = H_0 + H_1 + H_2 + \ldots,
\]

where \( H_0 \) defines the ground state, \( H_1 \) describes the dynamics of a single particle, \( H_2 \) the pure interaction of two particles and so on. Using the notation of second quantization in real space these operators can be written as

\[
\begin{align*}
H_0 &= E_0 \mathbf{1} \\
H_1 &= \sum_{i,j} t_{j,i} e_j^\dagger e_i \\
H_2 &= \sum_{i_1,i_2,j_1,j_2} t_{j_1,j_2,i_2,i_1} e_{j_1}^\dagger e_{j_2}^\dagger e_{i_2} e_{i_1} \\
&\quad \vdots \\
H_n &= \sum_{i_1, \ldots, i_n, j_1, \ldots, j_n} t_{j_1, \ldots, j_n, i_1 \ldots, i_n} e_{j_1}^\dagger \cdots e_{j_n}^\dagger e_{i_1} \cdots e_{i_n}. 
\end{align*}
\]

The creation and annihilation operators \( e_i^\dagger \) and \( e_i \) are local operators that act only on a single site \( i \) of the lattice and leave all other sites unchanged.

To compute operators on finite graphs for the linked cluster expansion the operators have to be of finite range and cluster additive [42, 46], where the concept
2.3 Linked Cluster Method

of cluster additivity is a generalization of the extensiveness required earlier for the high-temperature expansion. To understand this concept, we take a look at an arbitrary operator $O^C$ that acts only on the Hilbert space $\mathcal{H}^C$ spanned by the sites of graph $C$. The corresponding global operator $O$ acting on the whole Hilbert space $\mathcal{H} = \mathcal{H}^C \otimes \mathcal{H}^C$ is then given by

$$O := O^C \otimes 1^C,$$  \hspace{1cm} (2.42)

where $\overline{C}$ denotes all sites of the lattice which are not part of the graph $C$. An operator $O^C$ defined on a disconnected graph $C = A \cup B$, composed of the graphs $A$ and $B$, $A \cap B = \emptyset$, is cluster additive if it fulfills

$$O^C = O^A \otimes 1^B + 1^A \otimes O^B.$$  \hspace{1cm} (2.43)

For the n-particle irreducible Hamiltonians $H_n$ this is obviously true, since $A$ and $B$ do not share any edge and the creation and annihilation operators only act on single sites and hence $e_i^{(1),C}$ defined on a disconnected graph $C$ can be identified with $e_i^{(1),A} \otimes 1^B$ for $i \in A$ and $1^A \otimes e_i^{(1),B}$ for $i \in B$. For $H_1$, to give a simple example, this means

$$H_1^C = \sum_{i,j \in C} t_{ji} e_j^\dagger e_i$$

$$= \sum_{i,j \in A} t_{ji} e_j^\dagger e_i + \sum_{i,j \in B} t_{ji} e_j^\dagger e_i$$

$$= H_1^A \otimes 1^B + 1^A \otimes H_1^B.$$  \hspace{1cm} (2.44)

Hence $H_n$ is cluster additive. This also holds for the effective Hamiltonian $H_{\text{eff}}$, because $H_{\text{eff}}$ is a sum of cluster additive $H_n$ terms.

The remaining question is how to gain the n-particle irreducible $H_n$-operators from our effective Hamiltonian $H_{\text{eff}}$. In principle the answer is quite simple, but one has to be aware of cluster additivity: In the perturbative $T=0$ expansion, we apply the effective Hamiltonian on a state on the graph $A$ containing the desired number of particles and obtain the quantity

$$a_{j_1 \ldots j_n, i_1 \ldots i_1}^A = \langle j_1, \ldots, j_n | H_{\text{eff}}^A | i_1, \ldots, i_n \rangle.$$  \hspace{1cm} (2.45)

The problem is that $a_{j_1 \ldots j_n, i_1 \ldots i_1}^A$ is not the matrix element of the irreducible $H_n$. Instead it is the matrix element of $H_{\text{eff}}^A|_n$, which is $H_{\text{eff}}$ on graph $A$, restricted to $n$ particles. In contrast to $H_n$, $H_{\text{eff}}|_n$ contains contributions of all sectors with $m < n$ particles. In other words, even if $H_{\text{eff}}$ conserves the number of particles $n$ and only moves the particles on the lattice, not only the pure $n$-particle interaction is taken into account, but also underlying dynamics of less particles, e.g. the single particle
dynamics for each particle. Another problem is, that the operator $H_{\text{eff}}|n$ is not cluster additive and thus does not yield results in the thermodynamic limit. This is easy to see if one counts the number of particles in the disconnected graph $C$. According to the formula

$$H_{\text{eff}}^C|n = H_{\text{eff}}^A|n \otimes 1^B + 1^A \otimes H_{\text{eff}}^B|n$$

(2.46)

does not yield results in the thermodynamic limit. This is easy to see if one counts the number of particles in the disconnected graph $C$.

To obtain the matrix elements of the irreducible operators $H_n$ the idea is to subtract all contributions of $m < n$ particles,

$$H_0|0 = H_{\text{eff}}|0$$
$$H_1|1 = H_{\text{eff}}|1 - H_0|1$$
$$H_2|2 = H_{\text{eff}}|2 - H_0|2 - H_1|2$$
$$\vdots$$
$$H_n|n = H_{\text{eff}}|n - \sum_{i=0}^{n-1} H_i|n$$

(2.47)

as $H_n|m = 0$ for $n > m$ due to normal ordering. This yields a recursive set of equations to calculate the irreducible amplitudes $t_{j_1,\ldots,j_n,i_1,\ldots,i_1}$,

$$t_{j_i}^A = a_{j_i}^A - E_0^A \delta_{j_i}$$

(2.48)

$$t_{j_1,j_2,i_1,i_1}^A = a_{j_1,j_2,i_1,i_1}^A - E_0^A \delta_{j_1,i_1} \delta_{j_2,i_2} - E_0^A \delta_{j_1,i_1} \delta_{j_2,i_2} - E_0^A \delta_{j_1,i_1} \delta_{j_2,i_2} - E_0^A \delta_{j_1,i_1} \delta_{j_2,i_2}$$

(2.49)

where $E_0^A = \langle 0|H_{\text{eff}}^A|0 \rangle$ is the ground state energy of graph $A$. It is important that all matrix elements leading to the desired $t_{j_1,\ldots,j_n,i_1,\ldots,i_1}$ are calculated on the same graph. The irreducible amplitudes can then be used for the linked cluster expansion.

### 2.4 Series Analysis

The series obtained through high-temperature or perturbative series expansions have a very limited convergence radius in many cases. In some cases they are only valid in the direct vicinity of the infinite temperature $\beta = 0$ or the unperturbed $x = 0$ limit, respectively. The goal of series analysis is to extract useful information...
2.4 Series Analysis

on the underlying function of the series, and the physical quantity it describes, beyond those limits by extending the region of validity of the approximation and in some cases incorporating known features of the underlying function. The other core objective of series analysis is error estimation on the extracted quantities. For the bare series, with only a few coefficients available, even estimating the convergence radius rarely yields reliable results with standard methods. With current series analysis methods, the error estimation involves subtle evaluation of the results without a standard recipe. The resulting error bars are not comparable with statistical errors, but are rather qualitative estimates of the trustworthiness of the approximants for a given region. Which is why, how A. J. Guttmann and I. Jensen [47] put it, “often unfortunately they reflect the optimism of the investigator in the quality of his/her investigations!” If possible, it is good practice to study the series with various methods to get a comprehensive picture. Series analysis is a broad field on its own, and a full review is beyond the scope of this thesis. Here, we give an overview over the methods employed in presented studies. For a more detailed discussion we refer the reader to comprehensive reviews by A. J. Guttmann [48] and G. A. Baker [49].

2.4.1 Padé Approximation

A common method to extrapolate a series beyond the convergence radius and to provide a rough error estimate are Padé approximants [48, 50]. Padé approximants, sometimes also referred to as Padés, are a simple analytical continuation of the power series, which, in contrast to the bare series, can represent simple poles in the complex plane and therefore may better approximate the original function. A Padé approximant to a function $f(x)$ is a rational function

$$P[L, M](x) = \frac{a_0 + a_1 x + \cdots + a_L x^L}{b_0 + b_1 x + \cdots + b_M x^M}$$

(2.50)

of numerator degree $L$ and denominator degree $M$, whose Taylor series at $x = 0$ is identical to the Taylor series of the function $f(x)$ up to $O(x^{N+1})$, where $N = L + M$. Here we excluded the possibility of $f(x)$ having a pole at $x = 0$, also allowing us to make the conventional choice $b_0 = 1$ without further loss of generality. For each combination of $L$ and $M$ these conditions uniquely define the remaining real coefficients $a_i$, $b_j$. Due to their form, Padés approximate rational and meromorphic functions especially well. In case of a rational underlying function $f(x)$ they may even give the exact result for sufficiently high orders. In general, several theorems concerning the existence and convergence of Padé approximants $P[N, N - L]$ are known [48]. While they provide the fundamental justification for the use of Padé approximants from discussing the limit $N \to \infty$, they have no further practical implications for the analysis of our series, where only a few coefficients are available.
Series Expansions

For error estimation from Padé approximants various heuristic approaches exist, but as mentioned earlier, none of them provides rigorous error bars [47, 51]. A simple, yet effective, method is to inspect the behavior and spread of the Padé approximants \( P[L, N - L](x) \) for all \( 0 \leq L \leq N \) with increasing order \( N \). Approximants are trusted as long as most approximants of a given order \( N \) agree well. Higher confidence is usually put on balanced approximants, where the numerator and denominator are of roughly same degree [48]. As the sum of the numerator and denominator degree \( L + M \) corresponds to the order \( N \) of the series expansion in our application, the highest order approximants are of particular interest. Compared to the bare series, this method of error estimation adds almost no additional knobs to tune, keeping the investigator’s bias minimal. As it is also easily understood, most of our results were evaluated with this method. Another method relies on variable transformations and is discussed in section 2.4.3.

When interpreting Padé approximants, special attention has to be paid to the pole structure of the approximant in the complex plane. In physics applications singularities of the underlying function \( f(x) \) and its approximants on the real axis are often expected due to phase transitions. This forms a natural domain for the variable \( x \) on the real axis and defines a physical disc \( |x| \leq r_c \) in the complex plane, where \( r_c \) is the distance of the physical singularity from the origin. Approximants \( P[L, M] \) exhibiting a close root-pole pair within this physical disc are considered defective, since the pair would virtually cancel and add no further information compared to a Padé approximant of lower order. As these defective Padé approximants are very similar to their lower order relatives, they may suggest a false impression of convergence.

In general, Padé approximants are a very good starting point for an analysis, as they approximate most functions reasonably well, are easy to compute and offer direct access to the full graph of the approximant \( (x, y = P[L, M](x)), x \in \mathbb{C} \), which renders them ideal for fitting to a known function or experimental data.

### 2.4.2 DLog-Padé Approximation

For critical phenomena, where algebraic singularities play a central role, a variant of Padé approximants has proven very successful. Inspired by the expected power-law behavior close to a critical point

\[
f(x) \simeq \frac{A}{(x - x_c)^\gamma},
\]

(2.51)

where \( x_c \) is the critical point and \( \gamma \) is the associated so-called critical exponent, the idea is to create a Padé approximant on the first derivative of the logarithm of the function [52]. Close to the critical point this derivative is expected to behave
as
\[
\frac{d}{dx} \ln f(x) = \frac{f'(x)}{f(x)} \approx -\frac{\gamma}{x - x_c},
\] (2.52)
a functional form ideal for approximation with Padés. The resulting approximants are the DLog-Padé approximants \(P_{\text{DLog}}[L, M]\) of \(f(x)\). By construction DLog-Padé offer a convenient way to obtain estimates for the location of the critical point \(\hat{x}_c\) and the corresponding critical exponent \(\hat{\gamma}\), which can be determined from the roots of the denominator polynomial and its residue
\[
-\hat{\gamma} = \lim_{x \to \hat{x}_c} (x - \hat{x}_c) P_{\text{DLog}}[L, M](x),
\] (2.53)
respectively.

While DLog-Padé are very powerful tools for critical quantities, they are limited to positive definite functions and cannot handle additive analytic terms \(f(x) + C(x)\) [53, 54]. This also implies that confluent singularities close to the physical singularity \(x_c\) may slow down the convergence of the Padé approximants significantly and result in poor estimates from the given approximant [48]. Therefore, approximants with additional singularities close to \(\hat{x}_c\) should be discarded as defective as well. Furthermore, they are not well suited for fitting data on \(f(x)\), as estimates
\[
\hat{f}(x) = \exp \left( \int_0^x P_{\text{DLog}}[L, M](x')dx' \right)
\] (2.54)
on this function require the approximant to be integrated for each data point \((x_i, y_i = f(x_i))\).

When applying a DLog-Padé approximation to a series, note that one order is “lost” due to the derivative: given the \(N\)th order Taylor series of the function \(f(x)\), the degrees of the numerator and denominator of the DLog-Padé approximants \(P_{\text{DLog}}[L, M]\) are restricted to \(L + M \leq N - 1\).

### 2.4.3 Argument Transformations

Often Padé approximants are constructed for an argument transformed function \(f(x \to \phi(x))\) instead of \(f(x)\). A popular choice for the transformation are Euler transformations
\[
x \to \frac{ax}{1 + bx},
\] (2.55)
as diagonal Padé approximants \(P[L, L]\) are invariant under such transformations. This interesting feature implies that the Padé approximant at point \(x\) is the same for any point \(z\) reachable by an Euler transformation, if a convergent Taylor expansion exists for any of those points. It is the main reason for putting more confidence on the most balanced (also called tridiagonal) approximants \(P[L - 1, L], P[L, L]\),

22
and $P[L+1, L]$. Euler transformations are commonly used to move spurious poles out of the physical disc and may recover some of the defective Padé approximants. For DLog-Padé approximants, Euler transformations can be especially useful, as the derivative logarithm does not commute with the variable transformation and the argument-transformed approximant may behave better than the original DLog-Padé [48].

Applying argument transformation may have subtle consequences. For example, the influence of the higher-order coefficients of a series on the Padé approximant may be substantially reduced, resulting in an approximant incorporating only information on the lower orders of the series. C. J. Pearce [55] developed a set of conditions, which was extended by A. J. Guttmann [48], a good transformation function $\phi(x)$ must fulfill. However, those rules do not form a universal recipe to construct a well suited transformation for any series at hand and a transformation performing well in one case may have unwanted side-effects in another. A good example is the transformation $x \to \tanh(cx)$, where $c$ is a tunable parameter. While the method was very successful for Heisenberg models [56], the same transformation was found to strongly suppress peaks in a study on the $t$-$J$ model [39].

As mentioned before, variable transformations were also proposed for error estimation. The idea of the method, suggested by R. R. P. Singh and R. L. Glenister [51], is to generate a large number of Padé approximants through an argument transformations and taking the median and quartiles as the central estimate and error bars, respectively. The set of Padé approximants is built from a parameter-dependent transformation

$$x \to \phi_c(x),$$

(2.56)

where a large number of values for parameter $c$ yields a large number of different transformations, each resulting in a different approximant. The median and quartiles of those approximants constitute the estimate for the approximated function $f(x)$ and its errors. However, the method has a major flaw: it is unclear how to obtain a reasonable sample of transformation functions. If only transformation functions of a single functional form are considered, e.g. $\phi_c = \tanh(cx)$, the resulting approximants are highly correlated and likely subject to an artificial bias introduced by the transformation. Furthermore, the probability distribution chosen to sample the parameter $c$ has a substantial effect on the estimate as well as its errors, but there exist no rules on the properties of a sensible distribution.
2.4 Series Analysis

2.4.4 Multivariate Approximation

A straightforward generalization of Padé approximants to multivariate functions, e.g. \( f(x, y) \), are Canterbury approximants

\[
C[L_1, L_2, M_1, M_2](x, y) = \frac{\sum_{k=0}^{L_1} \sum_{l=0}^{L_2} a_{k,l} x^k y^l}{\sum_{m=0}^{M_1} \sum_{n=0}^{M_2} b_{m,n} x^m y^n},
\]  

(2.57)

where similarly to the univariate Padé approximants, \( L_1, L_2 \) are the degrees of the numerator and \( M_1, M_2 \) are the degrees of the denominator. Often only the diagonal approximants \( C[L, M] = C[L, L, M, M] \) are considered. While the idea to extend the method of Padé approximants to a multivariate form seems promising, in practice, Canterbury approximants mostly yield inferior results compared to fixing all but one variable before constructing a single-variable Padé approximant \([57]\). Therefore, we do not pursue the method of Canterbury approximants further, but analyze our multivariate series by univariate Padés after reducing the number of free variables.
Chapter 3

Graph Isomorphisms and Embeddings

The linked cluster method for the high-temperature series expansion, as well as, the perturbative series expansions at \( T = 0 \) involves four stages:

1. find all contributing connected graphs
2. compute the extensive quantity \( A^{(g)} \) for each graph
3. perform a subcluster-subtraction to obtain irreducible weights \( w^{(g)} \)
4. embed the graphs in the lattice and sum up the weights accordingly.

Three of those four stages involve important graph theoretical questions, namely

- Given two graphs, are those graphs isomorphic?
- Given two graphs, is one of those graphs embeddable in the other?
- Given a huge graph, what are the embeddable subgraphs with up to \( N \) edges?
- Given a small graph, what is the set of embeddable subgraphs?

In this chapter we lay out the algorithms we employed for solving those questions. They were developed in collaboration with Lukas Gamper.

Before we elaborate on the problems and their solution, let us briefly introduce a few basic graph theoretical definitions.

**Definition 1.** A graph \( g = (V, E) \) is a set of vertices \( V = \{1, \ldots, M\} \) and a set of edges \( E = \{e_1, \ldots, e_N\} \) connecting the vertices. In the context of lattice models, vertices and edges are also commonly referred to as sites and bonds, respectively.
3.1 Graph Isomorphisms

**Definition 2.** An undirected graph is a graph in which edges do not have a direction, i.e. an edge $e = (v_1, v_2)$ connecting vertices $v_1$ and $v_2$ is identical to the edge $e' = (v_2, v_1)$ connecting the vertices in the opposite order.

**Definition 3.** A simple graph is an undirected graph, where a pair of vertices can share at most one edge and edges connecting a vertex to itself are disallowed.

**Definition 4.** A connected graph is a graph $g$ where any vertex $v_1 \in V$ can be reached by any other vertex $v_2 \in V$ by following a path along the edges of the graph. Splitting a connected graph into two parts would cut at least one edge. A disconnected graph is a graph that is not connected.

**Definition 5.** A colored graph, has either colored vertices or colored edges or both, where the color is an intrinsic property or type of the vertex or edge, which distinguishes them from vertices and edges of other colors.

**Definition 6.** The degree (or order) $\deg(v, U)$ of a vertex $v$ with respect to a set of vertices $U$ is the number of edges connecting the vertex $v$ to any of the vertices $u \in U$. If no set $V$ is given explicitly, the set of the vertices of the graph $V_g \ni v$ is implied $\deg(v) = \deg(v, V_g)$, which is the number of edges incident on the vertex.

**Definition 7.** A subgraph $h$ of a graph $g$ is a graph whose set of vertices $V_h$ and set of edges $E_h$ are subsets of the respective sets $V_g, E_g$ of graph $g$. In return, $g$ is called supergraph of $h$.

**Definition 8.** A (weak) embedding of a graph $h$ in a graph $g$ is a mapping of all edges $E_h$ of graph $h$ to the vertices $E_g$ of graph $g$, such that all vertices $V_h$ of graph $h$ are uniquely mapped to vertices $v_g \in V_g$ of graph $g$.

The linked cluster expansion, also known as connected graph expansion, relies on simple connected graphs only. Thus we restrict our discussion to graphs of this kind and imply these properties for all graphs unless explicitly stated otherwise.

The algorithms described in this chapter were implemented in the graph library of the ALPS project (see appendix A).

### 3.1 Graph Isomorphisms

In the linked cluster method (sec. 2.3) we split the calculation of a property $A$ on the lattice into independent calculations of contributions of a large set of small embeddable graphs. Since we assume the contribution of the graph to be independent of its geometry in the lattice, but only to be depended on the edge and vertex structure of the graph, we save tremendous amounts of redundant computations by identifying identical graphs and computing their contribution only once. Thus
detecting if a graph is isomorphic to an already calculated graph is essential to the efficiency of the method.

As with many theoretical questions, the seemingly simple question if two graphs are isomorphic is surprisingly difficult and, in fact, poses a fundamental problem in graph theory. Interestingly, it belongs to the few problems in the complexity class NP, which are not known to be NP-complete or P for the general case [58, 59, 60].

The problem is given two graphs \( g = (V_g, E_g) \) and \( h = (V_h, E_h) \) is there a bijective mapping between the vertices \( V_g \) and \( V_h \) such that the edge-structure is conserved?

In case of colored graphs, we have to distinguish color-preserving isomorphisms, where the vertices have to be mapped onto vertices of the same color, while the edge-structure has to be preserved also with respect to the colors of the edges, and color-isomorphic isomorphisms, where vertex colors and edge colors may be mapped onto different colors.

The basis for our isomorphism detection is the McKay algorithm [61], an efficient algorithm to identify isomorphic uncolored or vertex colored simple connected graphs. For colored vertices the algorithm preserving the vertex colors in the mapping. We extended the algorithm to graphs with colored edges and, furthermore, included support for a generalized form of color-isomorphisms on the colored graphs. The “generalized color-isomorphism” allows to define color symmetry orbits and restricts the mapping of a color to colors of the same orbit.

3.1.1 McKay Algorithm

The McKay algorithm [61], which McKay also implemented in the software package nauty [62], decides if two graphs \( g, h \) are isomorphic by constructing canonical isomorph graphs \( C(g), C(h) \) which are identical for all isomorph graphs

\[
C(g) = C(h) \quad \text{iff} \quad g \text{ is isomorphic to } h. \quad (3.1)
\]

Here we follow the description of the McKay algorithm by Hartke and Radcliffe [63], which offer a comprehensive introduction from an application point of view. Unfortunately, we found a mistake in their example which will also address here. The original paper by McKay features the proofs of the theorems behind the algorithm, which were very helpful for the extension of the algorithm.

The key ideas behind the McKay algorithm are

- Define an ordering on the set of isomorph graphs, such that there is a unique least element under the ordering. This least graph is defined to be the canonical isomorph graph.

- Restrict the set of isomorph candidates by exploiting all degree information of the vertices as early as possible.
3.1 Graph Isomorphisms

Figure 3.1: Example graph with adjacency bit-matrix $E$.

- Build a search tree on the isomorph candidate graphs to find the least graph in the set.
- Prune the search tree with graph automorphisms found earlier in the search.

Given a simple connected graph $g = (V,E)$, we represent the graph by the set of vertices $V = \{1,2,\ldots,N\}$ and an upper-triangular adjacency bit-matrix $E$, which indicates if a pair of vertices is connected by an edge. The entry $E(i,j)$ is 1 if the vertices $i, j$ are connected by an edge and 0 otherwise. Figure 3.1 shows the graph and adjacency matrix of the example in [63]. The structure of the adjacency matrix depends not only on the graph structure, but also on the chosen order of vertices. The goal is to find an ordering, such that there is a unique canonical adjacency matrix for each set of isomorphic graphs. For this purpose we introduce an ordering $\preceq$ of isomorph graphs through the adjacency matrix, where we concatenate the rows and perform a lexicographical comparison of the resulting bit arrays, i.e. compared to a graph $h$

$$g \preceq h \iff E_g \leq E_h \text{ lexicographically.} \quad (3.2)$$

While this would suffice to define a canonical isomorph, the search space of candidates grows exponentially fast with the size of the graph, since the only way to find the least graph under this ordering is to construct the adjacency matrices of all possible permutations of vertices. One of the essential ideas of the McKay algorithm is to reduce the search space by putting additional conditions on the set of candidates using the degree information of the individual vertices.

Exploiting Degree Information

As the order of the vertices plays an important role for the adjacency matrices, we define the ordered partition of the vertex set.
Definition 9. An **ordered partition** $\pi$ of $V$ is a sequence $(V_1, V_2, \ldots, V_r)$ of non-empty sets, such that $\{V_1, V_2, \ldots, V_r\}$ forms a partition of $V$.

Furthermore, we define an order on the set of ordered partitions of the same vertex set $V$: a partition $\pi_2$ is **coarser** than a partition $\pi_1$ if the parts $W_k$ of $\pi_2$ can be obtained by forming unions of parts $V_i$ of $\pi_1$ while maintaining the overall order of the partition, i.e. if a part $V_i$ appears before another part $V_j$ of $\pi_1$ ($i \leq j$), the same has to hold for the parts $W_k \supseteq V_i$ and $W_l \supseteq V_j$ of $\pi_2$ ($k \leq l$).

The ordered partition allows us to classify the vertices and order them by their degree information.

Definition 10. An **equitable ordered partition** is an ordered partition $\pi = (V_1, V_2, \ldots, V_r)$, such that for all elements of the a part $v, w \in V_i$, the degree with respect to vertices of another part is the same $\deg(v, V_j) = \deg(w, V_j)$.

From an inequitable ordered partition $\pi$, such as the partition containing only a single part $(V)$, we would like to refine the parts by iteratively splitting them into parts based on the degree information of the vertices and find the coarsest equitable partition. Note that the coarsest equitable partition (or refinement) is unique up to the order of its parts. The procedure to obtain the coarsest equitable partition is the first important ingredient to the McKay algorithm and is called the “equitable refinement procedure”:

1. Initialize $\tau \leftarrow \pi = (V_1, \ldots)$.
2. Find the first pair $(V_i, V_j)$ in the ordered partition $\tau$ which violates $\deg(v, V_j) = \deg(w, V_j)$ for any $v, w \in V_i$.
3. If no pair is found, stop — $\tau$ is the coarsest equitable partition of $\pi$.
4. Replace $V_i$ in $\tau$ by the ordered partition of $V_i$, which are subsets $W_1, W_2, \ldots$ of $V_i$, each collecting the vertices of same degree $\deg(v, V_j) = \deg(w, V_j) \forall v, w \in W$, sorted by increasing degree $\deg(v, V_j)$.
5. Goto 2.

Note that a part may also be paired with itself $(V_i, V_i)$ in the second step. When deriving the coarsest equitable partition of a graph, one starts from a ordered partition of only a single part, and performs the equitable refinement procedure. In case of our example graph $g$, depicted in figure 3.1, the coarsest equitable partition is obtained after the first iteration:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\tau$</th>
<th>first violating pair $(V_i, V_j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${(1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9}$</td>
<td>$(V_1 = {1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9},\ V_1 = {1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}$</td>
</tr>
<tr>
<td>2</td>
<td>${(1 \ 3 \ 7 \ 9}, {2 \ 4 \ 6 \ 8}, {5}$</td>
<td>—</td>
</tr>
</tbody>
</table>

...
3.1 Graph Isomorphisms

While the procedure already exhausts all the degree information the vertices of the graph offer and induces an order the set of vertices based on the graph structure, it is in general not enough to uniquely define a canonical adjacency matrix. The remaining degree of freedom is the permutation of the vertices within each part of the equitable ordered partition. Compared to any permutation of vertices, the restriction introduced by classifying the vertices based on their degree information has already reduced the search space significantly. At this point the second important ingredient, the search tree, comes in.

Search Tree

Once the ordered partition incorporating all available degree information of the graph is found, the idea is to systematically explore the remaining possible permutations by artificially distinguishing a vertex within a non-trivial part of the ordered partition and continuing the equitable refinement procedure with the new degree information. For the artificial distinction of a selected vertex \( v \), one simply splits the non-trivial part \( V_i \) containing the vertex into a part consisting of the selected vertex and a part accommodating the remainder

\[
\pi = (V_1, \ldots, V_i, \ldots) \rightarrow \pi' = (V_1, \ldots, \{v\}, V_i \setminus \{v\}, \ldots).
\]

This scheme is repeated until the vertices are completely ordered, i.e. each part of the ordered partition contains only a single vertex. The partially ordered partitions having multiple vertices in at least one part constitute the nodes of the search tree, where each node has a child for each vertex of the first non-trivial part. The fully ordered partitions are the leaves of the tree.

Continuing on our example, the graph shown in figure 3.1, the equitable partition derived before \( \pi = (\{1\}, \{3 \ 7 \ 9\}, \{2 \ 4 \ 6 \ 8\}, \{5\}) \) form the root of the search tree, the first node is the equitable refinement of the partition where vertex 1 is separated from the first part:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>( \tau )</th>
<th>first violating pair ((V_i, V_j))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(({1}, {3 \ 7 \ 9}, {2 \ 4 \ 6 \ 8}, {5}))</td>
<td>( (V_3 = {2468}, V_1 = {1}) )</td>
</tr>
<tr>
<td>2</td>
<td>(({1}, {3 \ 7 \ 9}, {6 \ 8}, {2 \ 4}, {5}))</td>
<td>( (V_2 = {379}, V_3 = {68}) )</td>
</tr>
<tr>
<td>3</td>
<td>(({1}, {3 \ 7}, {9}, {6 \ 8}, {2 \ 4}, {5}))</td>
<td>—</td>
</tr>
</tbody>
</table>

The first child of the node splits the part \( \{37\} \), resulting in the equitable partition \((\{1\}, \{3\}, \{7\}, \{9\}, \{8\}, \{6\}, \{4\}, \{2\}, \{5\})\). As the vertices have now a definite order, this is the first leaf of the search tree. The full search tree for the example graph is shown in figure 3.2.

From the leaves of the search tree we obtain different canonical graph candidates by viewing each of the corresponding ordered partitions \( \pi \) as a permutation rule \( \sigma_\pi \), such that the rows and columns in the adjacency matrix appear in the same order as the corresponding vertices in the ordered partition.
Graph Isomorphisms and Embeddings

\[ \Pi = (\{1379\}\{2468\}\{5\}) \]

\[(\{1\}\{37\}\{9\}\{68\}\{24\}\{5\}) \quad \quad (\{3\}\{19\}\{7\}\{48\}\{26\}\{5\}) \quad \quad (\{2\}\{37\}\{1\}\{24\}\{68\}\{5\}) \]

\[(1|3|7|9|6|8|2|5) \quad (3|1|9|7|8|4|6|2|5) \quad (7|1|9|3|6|2|8|4) \quad (9|3|7|1|4|2|8|6|5) \]

Figure 3.2: Search tree of the McKay algorithm for the example graph shown in figure 3.1. The artificially distinguished vertex of each node is underlined. For better readability we changed the notation for the partitions with only trivial parts on the leaves. Note that this example tree differs from the example tree for the same graph shown in [63] due to a mistake in the equitable refinement.

For example, the first leaf \( \pi_1 = (1|3|7|9|6|8|2|5) \) represents the permutation \( \sigma_{\pi} = \{ 1 \mapsto 1, 3 \mapsto 2, 7 \mapsto 3, 9 \mapsto 4, \ldots \} \).

From all isomorph candidates generated by the search tree, the canonical graph is then the \( \succeq \)-least graph

\[ C(g) = \min_{\succeq} \{ g^{\sigma_\pi} : \pi \in \Lambda(g) \} , \quad (3.4) \]

where \( \Lambda(g) \) is the set of all leaves of the search tree for the graph \( g \).

Note that references [61, 63] chose the \( \preceq \)-greatest graph rather than the \( \succeq \)-least graph as canonical isomorph. Both conventions work equally well, as long as kept consistently.

Hartke and Radcliffe showed the example presented here also in their paper [63], but mistakenly selected the wrong pair of ordered partition parts in the second iteration of the equitable refinement procedure when building the first level of nodes of the search tree, which leads to a different set of isomorph candidates. The mistake becomes apparent when comparing the shown refinement example for the partition \( \pi = (\{1\}, \{379\}, \{2468\}, \{5\}) \).
3.1 Graph Isomorphisms

Automorphisms

While traversing the search tree, generating new leaves may reveal automorphisms of the graph. Whenever the permutations of two leaves $\pi$ and $\phi$ lead to the same adjacency matrix $E$, i.e. the same isomorph candidate, an automorphism $\gamma$ is discovered

$$g^\sigma_\pi = g^\sigma_\phi \Rightarrow g^\gamma = g, \quad \gamma = \sigma_\pi \sigma_\phi^{-1}. \quad (3.5)$$

If the search tree is generated in a depth-first fashion, the found automorphisms can be used to prune the remaining search tree. The subtree pruned due to the automorphism is related to an explored subtree by the automorphism and thus leads to the same candidate isomorphs. In more detail, consider a search tree where we already explored the subtree $A$ rooted in the node $a$ and start traversing the subtree $B$ rooted in node $b$, where $a$ and $b$ share the direct parent node $c$. If we find an automorphism $\gamma$ that relates a leaf $\pi$ of the subtree $A$ to a leaf $\phi$ of subtree $B$, the permutation of the automorphism $\gamma = \sigma_\phi \sigma_\pi^{-1}$ not only maps $\pi \to \phi$, but also maps the ancestor nodes $a \to b$, i.e. their ordered partitions, including the full search trees rooted in those nodes $A \to B$. The common parent node $c$ is fixed by the automorphism, as $\gamma$ maps vertices to vertices within the same partition. The subtree $B$ does not need to be explored any further and the search can proceed with the next child of $c$.

Furthermore, we obtain a complete set of generators of the automorphism group $\text{Aut}(g)$ while traversing the search tree. Since the initial ordered partition of the search tree distinguishes vertices only by their degree information, vertices in different initial partitions cannot be related by an automorphism. Therefore, every permutation $\gamma \in \text{Aut}(g)$, will map a leaf of the search tree $\pi$ to another leaf $\phi$ and will be detected. Pruning the search tree through automorphisms does not affect this conclusion, as only images of leaves related by already discovered generators will be removed.

Rather than collecting a list of generators, we accumulate the symmetry orbit

$$v^{\text{Aut}(g)} = \{v^\gamma | \gamma \in \text{Aut}(g)\} \quad (3.6)$$

of each vertex $v$. This can be achieved conveniently by the means of partitions of the vertex set $V$. Starting from discrete partition $O$ of the vertex set $V$, where each part contains only a single vertex, parts are merged as soon as a automorphism $\gamma$ maps a vertex $v$ of one partition $O_i \ni v$ to any vertex $v'$ of the other partition $O_j \ni v'$. This way a partition $O_i$ contains all vertices $v, v' \in O_i$ reachable by the action of the group of automorphisms discovered so far. When the search tree is completed and all the complete automorphism group $\text{Aut}(g)$ is found, the parts $O_i$ of $O$ are the orbits of the vertices in the respective part

$$O_i = v^{\text{Aut}(g)} \forall v \in O_i. \quad (3.7)$$
3.1.2 Colored Graphs

McKay’s canonical graph algorithm can be generalized to incorporate colored vertices and colored edges, i.e. vertices and edges of different kinds. In this section we presume that the colors are preserved under the isomorphisms, such that a vertex (or edges) of one color can only be mapped to vertices (edges) of the same color by the isomorphism.

For simplicity, we assume the sets of all vertex colors \( C_V \) and edge colors \( C_E \) present in the potentially isomorph graphs \( g \) and \( h \) are known and identical for the graphs. Furthermore, we assume there exists a strict order \(<\) on the set of colors and without loss of generality choose the set of \( K \) colors \( c \in C = \{1, 2, \ldots, k\} \).

Colored Vertices

Colored vertices are already covered by McKay’s original paper [61] and are easy to integrate. Since the search tree of the McKay algorithms does not mix parts of the initial partition \( \Pi \), vertices in different parts will not be considered for isomorphisms. Therefore, including not only the degree information of the vertices but also the color of the vertices in the initial partition is sufficient to find isomorphisms on graphs with colored vertices using the algorithm presented in the previous section. The only requirement is a well defined order of the parts of the initial partition \( \Pi \). This can be achieved by partitioning the set of vertices according to their color first \( (V_1, \ldots, V_k) \), in the order of the colors, and performing the equitable refinement procedure for each part \( V_1, \ldots, V_k \).

Colored Edges

Incorporating edge colors cannot be simply encoded into the initial partition, but the ordering \( \preceq \) yielding the canonical isomorph graph has to be extended.

For this purpose, we extend the uncolored adjacency bit-matrix \( E \) by a second bit-matrix \( C \), which captures the color information of the edges. The new color matrix \( C \) is a \( K \times M \) matrix, where \( K \) is the the number of colors and \( M \) the number of edges. If the \( i^{th} \) edge in the flattened adjacency matrix is of color \( l \) the matrix element \( C_{l,i} = 1 \), otherwise the element is \( C_{l,i} = 0 \). Combining the two structures we define a new ordering

\[
g \preceq h \iff (E_g, C_g) \leq (E_h, C_h) \text{ lexicographically}, \quad (3.8)
\]

where the lexicographical order is defined on the concatenated rows of the adjacency matrix and the rows of the color matrix. Due to the lexicographical ordering the color matrix only becomes relevant when comparing two graphs whose uncolored versions can not be distinguished \( E_g = E_h \). Since we assumed an order on
the set of colors, the color matrix $C$ has a well defined form in this case and will
either impose a strict order on the graphs $g$ and $h$ if the edges are of different
colors or reveal an automorphism, where $(E_g, C_g) = (E_h, C_h)$. Therefore, we may
still define the $\preceq$-least isomorph as the canonical isomorph.

Obviously encoding the color data in a matrix stores much redundant infor-
mation, as an edge $i$ can only be of a single color $k$. However, for small graphs
with only a few colors the bit-matrix allows for fast lexicographical comparison
on a computer and are fast to set up. For example, a graph with 20 edges of 3
different colors requires a 60 bit color matrix and can be compared using a single
64bit operation on modern CPUs.

Since the colors leave the plain degree information invariant, the initial partition
and the search tree also apply to the extended graphs and the additional fixed color
information will only inhibit some of the automorphisms present in the uncolored
case. The scheme will still yield all isomorph candidates.

However, there is one caveat: pruning the search tree based on found auto-
morphisms may break the isomorph candidate generation and may leave some
candidates undiscovered. The problem can be understood, when looking at the
common parent $c$ of the nodes $a$ and $b$ related by the discovered automorphism $\gamma$
(see page 32). In uncolored case the common ancestor $c$ is fixed by the automor-
phism, since the automorphism will only map vertices to other vertices in the same
part and all vertices within a part are equivalent in a sense, that they have the
same connections to all other parts. Therefore, permuting the vertices does not
have any effect. If the edges are colored vertices of the same part still share the
same number of connections to each other part, but now these connections may
be of different color. Thus, the vertices within the same part of $c$ are no longer
equivalent.

Automorphism based pruning of the search tree could be potentially be recov-
ered by considering the edge colors in the equitable refinement procedure. For our
purposes the algorithm performs well enough without pruning, since the graph for
the linked cluster method are small (10–20 edges) and result in search trees of
manageable sizes.

## 3.2 Graph Generation

The first step of a linked cluster expansion is determining all graphs embeddable
in the lattice of interest with up to $N$ edges. Here we present an efficient algorithm
to generate this list.

We start from an initial list of embeddable graphs, which is usually just a single
vertex without any edges and build trial graphs recursively by adding a new edge
between each pair of existing vertices and each of the existing vertices and a new
Graph Isomorphisms and Embeddings

<table>
<thead>
<tr>
<th>$N$</th>
<th>square</th>
<th>triangular</th>
<th>honeycomb</th>
<th>cubic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>12</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>28</td>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>28</td>
<td>72</td>
<td>12</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>68</td>
<td>198</td>
<td>22</td>
<td>83</td>
</tr>
<tr>
<td>9</td>
<td>156</td>
<td>590</td>
<td>45</td>
<td>206</td>
</tr>
<tr>
<td>10</td>
<td>399</td>
<td>1817</td>
<td>88</td>
<td>555</td>
</tr>
<tr>
<td>11</td>
<td>1012</td>
<td>5886</td>
<td>183</td>
<td>1517</td>
</tr>
<tr>
<td>12</td>
<td>2732</td>
<td>19753</td>
<td>389</td>
<td>4407</td>
</tr>
<tr>
<td>13</td>
<td>7385</td>
<td>68309</td>
<td>842</td>
<td>13005</td>
</tr>
<tr>
<td>14</td>
<td>20665</td>
<td>242352</td>
<td>1855</td>
<td>39931</td>
</tr>
<tr>
<td>15</td>
<td>58377</td>
<td>878568</td>
<td>4162</td>
<td>125225</td>
</tr>
<tr>
<td>16</td>
<td>168119</td>
<td>9445</td>
<td>403879</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>488771</td>
<td>21694</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>50355</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Number of embeddable graphs with exactly $N$ edges for various lattice geometries.

additional vertex. If multiple vertex or edge colors are possible, the recursion will try to add an edge or vertex of each color to the graph.

In each recursion step, the generated trial graph $g$ is first checked on isomorphisms against a list of already seen graphs using the canonical isomorph graph. If the trial graph has not been seen before the algorithm tries to find a possible embedding of the graph into the lattice using a backtracking approach. As this procedure is rather expensive for a large lattice, the trial graph has to pass two preliminary tests to detect non-embeddable graphs early:

1. the maximum degree $\Delta(g)$ of trial graph has to be compatible with the maximum degree of the lattice $\Delta(L)$. This rules out any graph which has more edges connected to a single vertex than any single vertex of the lattice can host.

2. the trial graph is checked against a list of smaller graphs which have passed the preliminary tests, but were found not to be embeddable in the end. Is any of those graphs embeddable in the trial graph, the trial graph itself will not be embeddable either.
Has the graph passed those preliminary tests the backtracking test will take place. Depending on its outcome the graph will then be added to either the list of embeddable graphs or the list of non-embeddable graphs used in the preliminary testing procedure.

The recursion will stop, if adding a new edge would yield a graph with $N + 1$ edges. Table 3.1 shows the number of embeddable graphs with $N$ edges generated by the algorithm for various lattice geometries.

3.3 Summary

In this chapter we introduced the graph algorithms which are essential for an efficient implementation of the linked cluster expansion, namely the detection of isomorph graphs and the generation of all non-isomorph embeddable graphs for a given lattice. We explained the graph isomorphism algorithm by McKay to find a canonical isomorph graph for any given graph, which also identifies automorphisms and thereby classifies the vertices of the graph based on their structural function. Furthermore, we showed how to treat graphs with colored vertices and generalized the McKay algorithm to graphs with colored edges.

Using the canonical isomorph algorithm, we presented an algorithm to generate all non-isomorphic graphs with up to $N$ edges embeddable in a given lattice.
Chapter 4

VLI — A Library for Large Integers and Polynomials

4.1 Introduction

High-temperature series expansions and ground state series expansions rely heavily on operations on symbolic polynomials in multiple variables with arbitrary precise coefficients. For high-temperature series expansions, the computational hot spot of the method are inner products of large vectors of such symbolic polynomials. The coefficients of the polynomials can be represented as integers whose maximal value can be determined before the actual calculation. For our application the required size of the integer to store this maximal value does not exceed 512 bits. Since the method is computationally expensive and offers multiple levels of exploitable parallelism, we require optimized kernels for these operations to make optimal use of today’s supercomputers and not waste valuable resources.

Apart from series expansions, integers offering a higher precision than the 64bit integers natively implemented on most current CPUs, have a variety of applications ranging from modern cryptography [64] implemented in our internet browser to scientific computer algebra programs (e.g. [65, 66]). Many multiple precision libraries, like the popular GNU Multiple Precision Arithmetic Library (GMP) [67] or the Number Theory Library (NTL) [68], provide an arbitrary precise integer class, which is dynamic in size and optimized to cover a wide range of integer sizes up to several thousand bits. Since the size of the integers in our application ranges only from 128 bit to 512 bit and the maximal size is known beforehand, the dynamic size poses a significant overhead and leaves room for more optimized implementations. The Number Theory Library (NTL) also offers a polynomial class with large integer coefficients. However, it does not support multivariate polynomials.
4.2 Large Integers

While the existing libraries may be optimized for common CPU architectures, current trends in supercomputing require applications to exploit massively multi-threaded GPU accelerators to benefit from their enormous computational capacity. Previous studies successfully explored polynomial multiplications on GPUs using FFT methods [69, 70, 71]. Here we are interested in the inner product of vectors of polynomials, which offers an additional level of parallelism. In order to meet the special requirements of the GPU on the memory layout the different components (vector, polynomial, high precision integer) need to be closely connected in order to interleave them for efficient parallel kernels. As this close connection is almost impossible if one of the components is provided by an external library, we decided to develop our own library to achieve best interoperability and not be restricted by external design decisions. While maintaining this high interoperability, the high precision integer part and the symbolic polynomial part of the library are designed in such a way that they can also be used independently. In the following sections will show key features and implementation decisions of these components and benchmark the library against a GMP solution.

The library was developed in collaboration with Timothée Ewart.

4.2 Large Integers

The library provides a template class \texttt{integer}<n> to represent signed integers with a fixed size of \(128 \leq n \leq 512\) bits. We implemented all standard integer operations starting from basic arithmetic operations, comparison operations to bit operations, like bit-shift or bit-wise logical operations. This way objects of the class can be used like the regular C++ type \texttt{int} in most cases. In addition to these standard integer operations we also provide a multiply-add function and an extended multiplication which doubles the number of bits when two integers of the same size get multiplied.

The data of the integer is stored in an array of 64 bit unsigned integer segments, where we use a standard two-complement representation of the number. Due to the fixed size of the integer, we are able to allocate the memory for the data on the stack. This way we save expensive heap allocations without implementing a manual memory pool management. For the basic arithmetic operations like addition and multiplication we use standard schoolbook algorithms, since the integers are too small for Toom [72] or FFT based approaches like the Schönhage-Strassen algorithm [73]. An exception is the 256 bit multiplication on the GPU where we use the Karatsuba algorithm [74] as we will explain later. We implemented the schoolbook algorithms in optimized assembly code to make optimal use of the hardware features that are not accessible from C++, like the carry addition \texttt{adcq} or the 64 bit multiplication \texttt{mulq} which calculates the low and high 64 bit part of
the product on the x86-64 architecture. Low-level programming methods for high-precision integer arithmetic and assembly tips can be found in the famous book “The Art of Assembly Language” [75]. Since writing assembly code can result in very long error-prone code, we generate many parts of the assembly code with the C++ preprocessor using the BOOST_PP library.

### 4.3 Polynomials

The second part of the library is a template class

```cpp
polynomial<CoeffType, Structure<N>, Var0, Var1, Var2, Var3>
```

for symbolic polynomials in 1 to 4 variables

\[
p(x, y, z, w) = \sum_{i,j,k,l} c_{ijkl} \cdot x^i y^j z^k w^l
\]

(4.1)

having either a “dense” \(i,j,k,l \leq N\) or a “triangular” \((i + j + k + l \leq N)\) structure. The truncation order \(N\) is fixed at compile time. The coefficients \(c_{ijkl}\) of the polynomial may have any C++ type supporting basic arithmetic operations. The polynomial itself supports additions, subtractions and multiplications with other polynomials or monomials. We also provide a special multiplication function returning a polynomial with truncation order \(2N\), such that no terms are dropped. It is also possible to mix and match polynomials having different sets of variables. The coefficients will be automatically mapped to the corresponding symbols during compile time. All operations call free hook functions with default implementations for all coefficient types. These functions may be overloaded by the user to provide optimized implementations for specific coefficient types. The user may for example add a specialized implementation for `float` using Streaming SIMD Extension intrinsics without touching the polynomial class itself. In addition to the general default implementations of the operations the library provides optimized routines for large integer coefficients `integer<n>`.

### 4.4 Optimized Inner Product

Since the hot spots of our main target application are inner products of large vectors with symbolic polynomials as components, where the coefficients of the polynomials are large integers, we implemented an optimized inner product function for this special case. All polynomials of the vector are assumed to have the same structure, the same truncation order \(N\) and the same `integer<n>` type as coefficients. The inner product will result in a polynomial of twice the order of
the original polynomials with coefficients of twice the width (integer\textsubscript{2n}) of the original coefficients. To employ today’s supercomputers as efficiently as possible the inner product is a hybrid CPU/GPU implementation, where the inner product is split into a part calculated on the CPU and a part calculated on the GPU. The ratio of these two parts can be set when building the library. It is also possible to deactivate the GPU part completely and compile a pure CPU version.

### 4.4.1 CPU Implementation

On the CPU we perform the inner product by element-wise polynomial-polynomial multiplications using the implementation of the polynomial. This operation is easily parallelized using OpenMP by splitting and distributing equal-sized chunks of the vectors among the available threads. Since all polynomials are of the same structure the work is well balanced between the threads.

The multiplication of two polynomials itself seems well suited for a SIMD implementation processing multiple coefficients in parallel. However, this is not advisable for large integers on current x86-64 architectures, even if the operations on the large integers are entirely independent. Using the Streaming SIMD Extensions (SSE) all kernels need to be based on 32 bit integer operations instead of 64 bit operations which are available for the sequential implementation. This reduction of segment size increases the number of required multiply instructions for a large integer multiplication by a factor of 4. The current SSE instruction set only supports an integer SIMD multiplication for two pairs of 32 bit numbers, each yielding a 64 bit result. Thus a SIMD implementation using SSE will be two times slower than our serial version using 64 bit integers. Note that we also neglected the carry bit propagation of the required additions. While the sequential version benefits from the hardware support for carry bit propagation, an SSE implementation needs to handle the carry manually.

The upcoming AVX2 instruction set will feature integer SIMD operations with twice the vector width of SSE. However, it will neither support 64 bit integer multiplications nor carry bit propagation. Therefore it will be at most as fast as our sequential version.

### 4.4.2 GPU Implementation

While on the CPU a SIMD approach is not promising, the highly parallel structure of the problem is well suited for GPU accelerators. In a nutshell, GPUs offer a hybrid between SIMD and massively multithreaded execution, where the GPU schedules thousands of threads in so-called warps. A warp is a group of usually 32 threads which are executed in a lockstep manner. We implemented the inner product in NVIDIA CUDA to exploit this powerful architecture. Our target hardware
is the NVIDIA Tesla K20X. We perform element-wise polynomial multiplications where each thread calculates one coefficient

$$c_{IJKL} = \sum_{i,j,k,l} a_{ijkl} \cdot b_{I-i,J-j,K-k,L-l}$$

(4.2)

of the product polynomials, where $a_{ijkl}$, $b_{i'j'k'l'}$ are the coefficients of the polynomials to be multiplied. Once all coefficients are calculated we perform a reduction over the result vector to obtain the final result of the inner product. This way we avoid race conditions and minimize the number of synchronization points. However, this method leads to a load imbalance since the number of terms in the sum (4.2) depends on the orders $I, J, K, L$ of the resulting coefficient. To overcome this problem we set up an execution plan before the actual calculation. Therefore we determine the number of terms of each coefficient of the product polynomial, sort them according to this number and assemble groups of 32 coefficients. Starting with the most expensive group the groups get scheduled on the available warps balancing the work among them (Fig. 4.1). Note that by sorting the coefficients we also minimize the work-load difference within a warp, such that the threads within the warp idle as little as possible until the other threads finished their lockstep calculation.

Once the `inner_product` function is called, the vectors to be multiplied are copied asynchronously from the host to the device. We store the data in the texture memory to take advantage of the texture cache which is optimized for 2D access. The intermediate results, i.e. the vector of the product polynomials, is written to the global memory. At this point we change the memory layout from the usual Array (vector) of Array (polynomial) of Structures (large integer) to an Array of Structures of Arrays where the data segments of the large integer coefficients are interleaved within the polynomial (Fig. 4.2), such that the least significant segments of all coefficients are contiguous in memory followed by the next more significant segments. This memory layout allows for an efficient coalesced access during the reduction at the end of the calculation. Once the reduction is completed the result is transferred from the device memory back to the host using the regular memory layout.

Like for the SSE approach on the CPU our kernels on the GPU rely on 32 bit unsigned integer arithmetic. However, on the GPU we were able to benefit from hard-wired addition with carry (`addc.cc`) and multiply-add with carry (`madc.cc`) operations using NVIDIA parallel thread execution (PTX) inline assembly. Contrary to the CPU where we used only schoolbook algorithms, we employ the Karatsuba algorithm [74] for the 256 bit integer multiplication on the GPU. Even though the method does not require less operations than the schoolbook algorithm it is advantageous on the GPU, because it uses less multiplications and more additions.
4.5 Benchmarks

Figure 4.1: (a) Number of contributions to each coefficient of the resulting polynomial (red line) sorted and split into tasks. (b) Load balancing of the tasks over the warps.

on the 32 bit segments, which have a 5 times higher throughput on the NVIDIA Kepler [76].

4.5 Benchmarks

4.5.1 Simple Integer Operations

We benchmarked the large integer part of the library on an Intel Sandy Bridge node and a Power7 node. We compared against the commonly used GNU Multiple Precision Arithmetic Library (GMP) version 5.1.1. The system configuration and compilation information is given in table 4.1. The libraries were tested and
Figure 4.2: Memory layout of a vector of polynomials with large integer coefficients. Each coefficient $c_{ijkl}$ consists of 64 bit segments $s$. On the CPU the coefficients are stored in nested arrays. Each coefficient is stored contiguously. For the intermediate results on the GPU, the coefficients are stored in an interleaved way grouping the segments according to their significance. This assures coalesced memory access. Note that on the GPU we will have segments of only 32 bit width, which doubles the number of segments compared to the 64 bit CPU version.

validated by the GMP benchmark. Comparing the achieved GMPbench scores to reference values on the GMP web page [67] for slightly different systems shows good agreement. The benchmarks and the VLI library were compiled with -O2 -m64. Comparing the performance of simple addition and multiplication operations (Tab. 4.2) our implementation for fixed size integers is between 10% and 302% faster for the addition and up to 150% faster for the multiplication. We obtain larger speed-ups for the addition of small high precision integers, since the overhead to manage the dynamic integer size in GMP is approximately independent of the integer size.

<table>
<thead>
<tr>
<th>CPU</th>
<th>Sandy Bridge</th>
<th>Power7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler</td>
<td>Intel Xeon E5-2670 (16 physical cores, 2.6 GHz)</td>
<td>Power7 720 Express (8 physical cores, 3.0 GHz)</td>
</tr>
<tr>
<td>GMP compile flags</td>
<td>-02 -fomit-frame-pointer -m64 -march=corei7</td>
<td>-03 -m64 -mtune=power7</td>
</tr>
<tr>
<td>GMPbench 0.2 score</td>
<td>41580 (multiply), 1245 (full score/Ghz)</td>
<td>26040 (multiply), 670 (full score/Ghz)</td>
</tr>
</tbody>
</table>

Table 4.1: Benchmark system configuration
Table 4.2: Performance of the GNU Multiple Precision Arithmetic Library and the VLI library for addition (+) and multiplication (×) in $10^6$ large integer operations/s. Mean and standard deviation from 100 runs.

<table>
<thead>
<tr>
<th>op.</th>
<th>Power7</th>
<th>SandyBridge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GMP</td>
<td>VLI</td>
</tr>
<tr>
<td>128 bit +</td>
<td>58.2 ±1.3</td>
<td>107 ±1.4</td>
</tr>
<tr>
<td>192 bit +</td>
<td>61.1 ±1.2</td>
<td>79.4 ±2.9</td>
</tr>
<tr>
<td>256 bit +</td>
<td>59.2 ±1.4</td>
<td>66.7 ±1.7</td>
</tr>
<tr>
<td>320 bit +</td>
<td>53.5 ±1.1</td>
<td>59.7 ±1.0</td>
</tr>
<tr>
<td>384 bit +</td>
<td>51.7 ±0.9</td>
<td>64.6 ±1.6</td>
</tr>
<tr>
<td>448 bit +</td>
<td>54.7 ±1.2</td>
<td>56.7 ±1.1</td>
</tr>
<tr>
<td>512 bit +</td>
<td>52.5 ±1.0</td>
<td>50.0 ±3.3</td>
</tr>
<tr>
<td>128 bit ×</td>
<td>31.2 ±0.4</td>
<td>103 ±2.7</td>
</tr>
<tr>
<td>192 bit ×</td>
<td>18.6 ±2.4</td>
<td>61.4 ±1.0</td>
</tr>
<tr>
<td>256 bit ×</td>
<td>16.0 ±2.2</td>
<td>34.9 ±0.5</td>
</tr>
</tbody>
</table>

Since our integer library is stack based we do not need expensive calls to malloc like GMP, which will give an additional speedup for the inner product, where multiple allocations are necessary.

### 4.5.2 Optimized Inner Product

The benchmarks for the inner product were again performed on the Intel Sandy Bridge node for the pure CPU version of the inner product. The GPU benchmarks were performed on Todi, a Cray XK7 with NVIDIA Tesla K20X GPUs and AMD Opteron CPUs, at the Swiss Center for Scientific Computing (CSCS). The test cases are inner products of two vectors of dimension 4096 with “dense” and “triangular” polynomials of order 1 to 14 with 128 and 256 bit integer coefficients. Since the inner product will double the order of the polynomial and the width of the large integer coefficients, the test cases will result in polynomials of order 2 to 28 with 256 and 512 bit integer coefficients, respectively.

### CPU Implementation

Figures 4.3a and 4.3b show the performance behavior of the inner product for polynomials with 128 bit integer coefficients for different truncation orders. Let us first focus on the pure CPU based inner products, which are represented by dashed lines. The naive implementation using the GNU Multiple Precision Arithmetic Library (GMP) and OpenMP performs rather poorly reaching $0.1 \cdot 10^9$ large integer operations per second for any polynomial type. As suggested before a major reason for the poor performance is the dynamic memory management of GMP. A VTune
Table 4.3: VTune Performance profile of the inner product using GMP on Intel Sandy Bridge for polynomials of order 10 in three variables.

<table>
<thead>
<tr>
<th>Function</th>
<th>Time [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>__gmp_default_reallocate</td>
<td>34</td>
</tr>
<tr>
<td>__gmp_default_allocate</td>
<td>14</td>
</tr>
<tr>
<td>vli::detail::inner_product_cpu</td>
<td>10</td>
</tr>
<tr>
<td>__gmpz_add</td>
<td>10</td>
</tr>
<tr>
<td>__gmpz_mul</td>
<td>9</td>
</tr>
<tr>
<td>__gmpn_mul_2</td>
<td>7</td>
</tr>
<tr>
<td>__gmpz_realloc</td>
<td>3</td>
</tr>
<tr>
<td>__gmpz_init</td>
<td>3</td>
</tr>
<tr>
<td>__gmpn_add_n</td>
<td>2</td>
</tr>
<tr>
<td>__gmpn_sub_n</td>
<td>2</td>
</tr>
</tbody>
</table>

Profile analysis (Tab. 4.3) reveals approximately 50% of the execution time is spent in memory allocation.

Using our optimized CPU based inner product, we reach a limit of $1.6 \cdot 10^9$ large integer OP/s for “dense” polynomials in two and three variables for almost any order. The performance of the optimized inner product for univariate polynomials remains below this limit and grows slowly with the order of the polynomial, because the amount of work per thread is too low and the computation is dominated by spawning the OpenMP threads. The performance behavior of the “triangular” polynomials is similar (Fig. 4.3b), however the upper limit depends on the number of variables of the polynomial. We achieve $1.1 \cdot 10^9$ large integer OP/s for polynomials in two symbolic variables, and only $0.9 \cdot 10^9$ large integer OP/s for three symbolic variables. This difference is due to the more complex index calculations required to map the “triangular” structure to the linear memory layout. In conclusion we managed to gain a speed-up of 13x for “dense” polynomials and up to 10x for “triangular” polynomials using our CPU based optimized inner product and large integer functions.

**GPU Implementation**

Using the optimized inner product only on GPUs yields an even higher performance (Fig. 4.3, solid lines). Note that the performance estimate includes the time needed for the data transfer between the host and the device. For very low orders as well as for univariate polynomials, the GPU is outperformed by the CPU implementation, since these calculations do not offer enough parallelism to saturate the available number of threads on the GPU. The performance, however, increases with increasing order until it reaches a plateau at $4.6 \cdot 10^9$ large integer OP/s for
4.5 Benchmarks

Figure 4.3: Comparison of the inner product with and without GPU accelerator. Inner product of “dense” polynomials (a and c) and “triangular” polynomials (b and d) up to 3 variables, with 128 bit (a and b) and 256 bit (c and d) coefficients. Size of the vector 4096. GMP gives similar results independent of the polynomial structure.

a 7th order “dense” polynomial in three variables (Fig. 4.3a). This increase is mainly due to the growing number of coefficients which allow us to employ more threads and eases the load balancing as discussed in the previous section. Beyond 13th order the performance decreases again, since the texture memory cache is saturated and the number of cache misses increases. For polynomials in two variables this threshold is not reached within our benchmark. The performance of the inner product for “triangular” polynomials increases less with increasing order in general, as the number of coefficients grows slower and the calculation of the
memory location is more complex than for “dense” polynomials. Compared to the
naive GMP solution we achieve speed-ups of up to 38x and 33x for “dense” and
“triangular” polynomials, respectively. This corresponds to an additional speed-up
of 3x with respect to our pure CPU inner product on the Sandy Bridge node.

The performance profile of the inner product for polynomials with 256 bit
integer coefficients (Fig. 4.3c and 4.3d) is very similar the 128 bit version. Since
the 256 bit integer multiplication requires approximately four times the number
of operations of the 128 bit integer multiplication, one would expect at most 1/4
of the large integer operation performance of the 128 bit operation. However,
the number of large integer operations per second is only reduced by a factor of
approximately 2/5. The reason for this non-linear behavior is twofold. First, the
load of the data is only twice as expensive for the 256 bit operation, since during
the operation all data is kept in registers and each number needs to be loaded only
once per large integer multiplication. This is true for the CPU and the GPU. The
second reason is architecture dependent. The CPU can employ instruction level
parallelism techniques more effectively, since the 256 bit multiplication contains
more independent operations than the corresponding 128 bit version. On the GPU
we profit from the Karatsuba algorithm for 256 bit multiplications trading slow 32
bit integer multiplications for additions, which have a much higher throughput on
the GPU. Thus the additional operations can be executed in a more efficient way.

4.5.3 Efficiency of the Optimized Inner Product

To estimate the efficiency of our GPU kernel we computed the number of 32 bit
integer operations per second (IOP/s) for the inner product with 128 bit integer
coefficients. We only considered the instructions of the polynomial-polynomial
multiplication and neglect the reduction at the end of the inner product. Since
to our knowledge there exist neither benchmarks nor official peak values for the
integer performance of the NVIDIA Kepler GK110, we calculate a theoretical limit
based on the instruction throughput of the 32 bit integer multiply-add (madc)
instruction [76], which is used almost exclusively in our multiplication kernel. The
theoretical maximum of the NVIDIA Tesla K20X is given by

\[
I_{\text{SMX}} \cdot f \cdot \mu \cdot r = 14 \text{SMX} \cdot 732 \cdot 10^6 \text{cycle/s} \cdot 32 \frac{\text{instr}}{\text{SMX} \cdot \text{cycle}} \cdot 2 \frac{\text{IOP}}{\text{instr}} = 655 \text{GIOP/s},
\]

(4.3)

where \(n_{\text{SMX}}\) is the number of streaming multiprocessors (SMX), \(f\) denotes the
clock frequency, \(\mu\) is the instruction throughput of the madc instruction and \(r\) is
the number of integer operations per instruction. Since GPUs are optimized for
single precision floating point arithmetic, the theoretical integer performance of
the NVIDIA Tesla K20X is just 655 GIOP/s, which is far less than the floating point performance of 3.95 TFLOP/s. The maximal performance of our implementation of $4.6 \cdot 10^9$ large integer OP/s in figure 4.3 corresponds to $4.6 \cdot 10^9 \text{OP/s} \cdot 32 \text{instr/OP} \cdot 21 \text{OP/instr} = 294 \text{GIOP/s}$ (32 bit). This number includes the time needed to transfer the data between the CPU memory and the GPU memory. In an independent measurement we determine the pure kernel performance to be 319 GIOP/s. We therefore reach 49% of the theoretical peak performance of the NVIDIA Tesla K20X.

4.6 Summary and Outlook

We presented a C++ library for efficient fixed high precision integers up to 512 bits and polynomials in 1 to 4 symbolic variables with an optimized inner product function. The purely CPU based high precision integer part of the library outperforms the GNU Multiple Precision Arithmetic Library (GMP) by a factor up to 4.2x for the integer addition and 2.5x for the integer multiplication. The optimized hybrid CPU-GPU inner product function for vectors of polynomials with high precision integer coefficients in pure CPU mode showed an excellent speed-up factor of up to 13x compared to a naïve solution using GMP and OpenMP. Using a single NVIDIA Tesla K20X we were able to push this speed-up factor to 38x. While the library is originally intended as a special purpose library for high-temperature series expansions, its modular design offers a high flexibility and renders the library also interesting to other applications in science and engineering.

We would like to extend the large integer class to allow for sizes up to 4096 bits, which will make it more attractive to applications in cryptography. The polynomial part of the library should allow for more complex structures, like individual truncation orders for each symbolic variable, and could be extended to support lazy evaluation. Until now the hybrid inner product function requires the library user to specify the ratio that is used to split the vectors to be multiplied between GPU and CPU when building the library. In future versions we would like to provide a tool to generate a look-up table based on a small benchmark to select the appropriate ratio automatically. Beside the optimized kernels for the x86-64, power64 and Kepler architectures we would also like to explore the new Intel Xeon Phi SIMD architecture, which supports carry propagation natively and might be a better candidate for integer arithmetic than current GPUs.

The library is released under the Boost Software License and available at http://www.comp-phys.org/vli/.
Chapter 5

High-T Series for the Heisenberg Model

5.1 Introduction

Quantum antiferromagnets in low dimensions are a major topic in condensed matter physics. The initial reason for intensive study of antiferromagnetic systems was the discovery of antiferromagnetic order in the copper oxide layers of the undoped parent compounds of high-temperature superconducting cuprates [77]. Since then, research on low dimensional antiferromagnetic structures has evolved into an independent field because these systems are strongly affected by quantum fluctuations and offer a great variety of exotic phases, like valence bond solids or quantum spin liquids [78]. From a theoretical perspective they can be described by Heisenberg models, which are, due to their simplicity and the many exotic phases they exhibit, one of the most important class of toy models to study quantum phase transitions [79]. Furthermore, they can also serve as well controllable environment to investigate more general phenomena like Bose-Einstein condensation [80]. There exist plenty of experimental realizations for quantum magnets, such as the undoped La$_2$CuO$_4$, NaTiO$_2$ [81] or superconducting organic molecular crystals [82], to name a few. A common problem is the connection of the experiments to the theoretical models, i.e. the determination of coupling constants of theoretical models for a given material [83, 84].

An easy link between experiment and theory can be established by high-order high-temperature series expansions for the microscopic models, since the method yields a symbolic high-temperature series of macroscopic quantities, such as the magnetic susceptibility, in the microscopic coupling constants of the model. As high-temperature series expansions provide results directly in the thermodynamic limit for translational invariant lattices, the finite order of the series remains the
5.2 Model

The spin-1/2 Heisenberg model is a lattice model where each site is occupied by a spin \( i \) spanning a local Hilbert space \( \mathcal{H}_i = \{ \uparrow, \downarrow \} \). The spins couple along bonds of the lattice and are governed by the Hamiltonian

\[
H = \sum_{\langle i,j \rangle} J_{i,j} \mathbf{S}_i \mathbf{S}_j,
\]  

(5.1)

where \( \mathbf{S}_i \) are the spin-1/2 operators acting on the spin \( i \), and \( J_{i,j} \) are coupling constants for each bond \( \langle i, j \rangle \) of the lattice. For the models considered, we only consider a small set of independent coupling constants \( J_{i,j} \in \{ J_1, J_2, \ldots \} \). Bonds with the same coupling constant are said to be of the same type. While our application can handle up to four different bond types, we here focus on a model with only two bond types, namely the Heisenberg model on an anisotropic triangular lattice depicted in figure 5.1. With two freely tunable constants \( J_1, J_2 \) the model covers three well known special cases: for \( J_1 = 0 \) the model decomposes into uncoupled spin chains, for \( J_2 = 0 \) the model is equivalent to the Heisenberg model on a square lattice, and for \( J_1 = J_2 \) it becomes an isotropic triangular lattice.
The high-temperature series we obtain are symbolic polynomials in the variables $\beta J_1, \beta J_2$. As the coupling constants remain freely tunable our series is valid for both the ferromagnetic ($J_1, J_2 < 0$) and the antiferromagnetic case ($J_1, J_2 > 0$). In our analysis of the results we focus on the latter, as this case leads to geometric frustration on the triangular lattice as not all bonds can be satisfied classically. This enhances quantum fluctuations and promises an interesting phase diagram. Furthermore, the geometric frustration of the model makes it hard to study the system with the otherwise successful method of Quantum Monte Carlo (QMC) due to the sign-problem caused by the frustration. Hence the model renders an ideal playground for our high-temperature series expansions.

5.3 Running the Application

As previously mentioned in chapter 3, the computation of a high-temperature series expansion involves four stages:

1. find all contributing connected graphs,
2. compute the desired quantity $A^{(g)}$ for each graph,
3. obtain the irreducible weights $w^{(g)}$ through subcluster-subtraction,
4. embed the graphs in the lattice and sum up the weights accordingly.

Similar to those four stages running our code for the spin-1/2 Heisenberg model on a given lattice involves three steps, each carried out by a dedicated application as shown in the respective example calls:

1. the graph generation, creating a list of all graphs embeddable into the desired lattice
   \[
   \text{graph_generator latticeparam.json graphlist.graphs}
   \]
2. the weight computation, calculating the contribution to the free energy series, magnetic susceptibility series and spin structure factor series for each graph
   \[
   \text{spin1_2_heisenberg_hte_compute_weights_2_variables 10 graphlist.graphs raw_heisenberg.series}
   \]
3. and finally the embedding step, performing the subcluster-subtraction and embedding into the lattice.
   \[
   \text{spin1_2_heisenberg_hte_reduce_embed_weights latticeparam.json graphlist.graphs raw_heisenberg.series reduced_heisenberg_series}
   \]
5.3 Running the Application

Listing 1: The parameter file for the graph_generator program used in our example.

In the following sections we explain these steps and the associated applications in more detail.

5.3.1 Graph Generation

The first program

```
graph_generator <parameterfile> <outputfile>
```

generates all embeddable graphs with up to $N$ edges for a lattice specified using the ALPS lattice library [1][86]. Starting from a single vertex it generates graphs by recursively adding edges, checking for already seen isomorphic graphs using a variant of the McKay algorithm [61, 63] and trying to embed these graphs in the lattice. The program takes two arguments:

- `<parameterfile>`, the name of a parameter file in JSON format, and
- `<outputfile>`, the name of the output file in which the found graphs are stored.

Listing 1 shows the parameter file used for the anisotropic triangular lattice.

The first parameter, "lattice", describes the lattice for which the embeddable graphs are generated. The lattices are provided by the ALPS lattice library which offers a big selection of predefined lattice and is easily extensible.

The parameters "type" and "L" are the name and the extent of the lattice and are passed through to the ALPS lattice library. As our example relies on a lattice which is not part of the standard ALPS lattice library, a custom library file is loaded via the "lattice_library" parameter. Note that "L" is not the size of the (infinite) lattice used in the calculations, but indicates a finite lattice large enough so that embedded clusters starting at the center do not extend to the boundary.
The next parameter, "num_edges", sets the upper limit on the number of edges of the generated graphs and should match the desired order of the series to be computed.

The last parameter tells the graph generator that all terms of the Hamiltonian, regardless of the edge type, are of the same form \((S_i S_j)\).

For the anisotropic triangular lattice depicted in figure 5.1 we found 4'821'837 embeddable graphs with \(n \leq 12\) edges.

### 5.3.2 Weight Computation

The weight computation is the most expensive part of the procedure. Due to the rapidly growing number of graphs for higher orders, as well as the exponentially growing Hilbert space with the number of sites in a graph, the computational effort grows exponentially with the desired order of the series. We provide applications for up to four independent coupling constants

1. spin1_2_heisenberg_hhe_compute_weights_1_variables
2. spin1_2_heisenberg_hhe_compute_weights_2_variables
3. spin1_2_heisenberg_hhe_compute_weights_3_variables
4. spin1_2_heisenberg_hhe_compute_weights_4_variables

Each application accepts the following three parameters

```
... [--suscept-only] <order> <graphfile> <outputfile>
```

where the first mandatory parameter <order> is the desired order of the expansion, followed by <graphfile>, the name of the file containing the list of graphs for which the weights should be computed, and <outputfile>, the name of the output file. For each graph the application will compute the series for the partition function, the susceptibility and the equal-time \(S_i^z S_j^z\) correlator for the static structure factor \(S(k)\). All results will be appended to the combined output file. If the structure factor is not of interest, the mandatory arguments can be preceded by the optional switch --suscept-only, telling the application to omit the calculation of the correlator series. This allows to focus on pushing the susceptibility to the highest possible order.

If the output file <combinedseriesdb> already exists, the program will only compute and append the weights of the missing graphs. Therefore the computations can be interrupted at any time and may be continued by re-executing the same command.

For short series the regular applications for desktop computers suffice. However, two to three additional orders may be achieved running the MPI-parallelized versions of the applications on a cluster or supercomputer. The MPI applications will group the graph contributions to be computed in tasks of a few graphs each.
and schedule those tasks across all available MPI processes. The applications, which have to be executed within an MPI environment, share the same command line arguments as their non-MPI counterparts, but take three additional optional parameters controlling the parallelization:

```
... [--suscept-only] <order> <graphfile> <outputfile> [threadssp]  
  [checkp_interval] [task_size]
```

- `threadssp` is the number of threads per MPI process (default: 16). In most cases a single MPI process per cluster node should be used, running one thread per CPU-core of the node.

- `checkp_interval` sets the interval of checkpoints in seconds (default: 30). At the end of each interval the completed computation tasks will be collected from all MPI processes and stored in the output database.

- `task_size` controls the size of a computation task, i.e. the number of graphs per computation task (default: 15). Note that the program will only save completed tasks, where all graph contributions have been calculated, to the database at the checkpoints. Large `task_size` and `checkp_interval` values may therefore result in dropping many computed contributions in incomplete tasks at termination of the program. Very small values, on the other hand, may cause considerable network traffic on the cluster.

If these optional arguments are not specified their default value will be used.

### 5.3.3 Subcluster-Subtraction and Embedding

Once the raw weights are calculated for all contributing graphs, the

```
spin_2_heisenberg_hete_reduce_embed_weights <parameterfile> <graphfile>  
  <combinedseriesdb> <seriesdb_prefix>
```

program will perform the subcluster-subtraction to obtain the irreducible weights and sum up those weights according to the embeddings of the respective graphs in the lattice. For this task the program requires four arguments:

- `<parameterfile>`, the name of the json file containing the lattice parameters,

- `<graphfile>`, the name of the file containing the list of contributing graphs for the lattice,

- `<combinedseriesdb>`, the name of the file with the combined raw weights for the partition function, the susceptibility and optionally the two-site correlator for all contributing graphs from the previous step, and
• `<seriesdb_prefix>`, a prefix for the intermediate files the program will generate.

From the combined raw weights file the program generates three files containing the full graph contributions before the subcluster-subtraction:

- `<sp>_raw_mBetaF.series` free energy ($-\beta F$)
- `<sp>_raw_suscept.series` susceptibility
- `<sp>_raw_SzSz_correlator.series` $S^z_i S^z_j$ correlator

and three files with the irreducible weights after the subcluster-subtraction:

- `<sp>_red_mBetaF.series` free energy ($-\beta F$)
- `<sp>_red_suscept.series` susceptibility
- `<sp>_red_SzSz_correlator.series` $S^z_i S^z_j$ correlator.

The prefix `<sp>` of the filenames is substituted by the value of the command line argument `<seriesdb_prefix>` mentioned above.

Once more, if the output file already exists any already calculated quantity will be taken into account and only missing weights will be appended. As the weights of the graphs are independent of the underlying lattice, the weight files may contain weights for graphs not contributing to the lattice. Once the subcluster-subtraction is completed and all irreducible weights are available, the program tries to embed all graphs from the graph list file into the lattice. When the embedding is completed it will print out the resulting series in Mathematica compatible format to standard-out.

### 5.4 Series Analysis

Analysis of the high-$T$ series is done in Mathematica, which allows us to easily perform symbolic manipulations and has many of the needed tools, most notably Padé approximants, built-in. Along with the high-temperature series expansion application code we ship two Mathematica packages\footnote{padeanalysis.m and paderesidualanalysis.m found in `/opt/lcse/share/lcse/data_analysis` after installation.} which complement the built-in tools and provide functions to simplify common tasks of the analysis. Here we show how to extract estimates for the coupling constants for the Heisenberg model from susceptibility measurements on Cs$_2$CuBr$_4$, using Padé approximants constructed from the high-temperature series.

#### 5.4.1 Extraction of Coupling Constants

The effective coupling constants of the Heisenberg model for a real quantum magnet can be estimated by fitting the Padé approximants of the magnetic susceptibility $\chi(T)$ to experimental data. As an example we illustrate how to determine the
5.4 Series Analysis

coupling constants of the Heisenberg model on an anisotropic triangular lattice for Cs$_2$CuBr$_4$. The full procedure can be found in the Mathematica notebook\textsuperscript{2} in the example folder of the code package.

In the beginning of the notebook we load the Mathematica packages shipped with the code and paste the high-temperature series $\chi/\beta$ copied from the output of the series expansion application

\begin{verbatim}
<< "padeanalysis.m"
<< "paderesidualanalysis.m"
chiOverBeta = 1/4 - 1/8*K + (*...*) - 1/4*B + (*...*)
\end{verbatim}

where the the symbolic variables B and K correspond to $\beta J_1$ and $\beta J_2$, respectively.

For the fit to experimental data, three free parameters of the susceptibility function $\chi(T)$ of our model need to be determined: the Heisenberg couplings $J_1$, $J_2$ and the $g$-factor. As the couplings $J_1$ and $J_2$ appear as non-linear parameters in the Padé approximants, we determine these parameters by a parameter scan on an equidistant grid, calculating the sum of squared residuals to the experimental data for each grid point.

The range for the grid is derived from the behavior of the system at very high temperatures, where the Curie-Weiss law

$$\chi(T) = \frac{C}{T_{CW} + T} \quad \text{(5.2)}$$

applies. Here $\chi(T)$ is the magnetic susceptibility per mole, $C$ is the Curie constant $C = N_A g^2 \mu_0 \mu_b^2 / 4 k_B$ and $T_{CW}$ is the Curie-Weiss temperature. From a fit to the experimental data we obtain a first estimate for the $g$-factor and the Curie-Weiss temperature, which relates to the coupling constants of our system through mean-field theory

$$T_{CW} = J_1 + J_2 / 2. \quad \text{(5.3)}$$

Depending on the temperature range considered, the fit of the Curie-Weiss law for Cs$_2$CuBr$_4$ yields $1.94 < g < 2.03$ and $3K < T_{CW} < 20K$ (not shown). To benefit from this first estimate the free parameter $J_2$ in our analysis is substituted by $T_{CW}$ according to eq. (5.3).

For the parameter scan we define an equidistant grid for $5K \leq T_{CW} \leq 20K$ and $0.1K \leq J_1 \leq 15K$ with a grid spacing of $0.1K$ for each parameter. The Mathematica packages, loaded before, offer a set of tools for this scan, which are controlled by a common parameter set. Such a parameter set is created with the \texttt{CreatePadeResidualLandscapesParameterSet} function from a list of Padé parameters $[L, M]$ and the ranges forming the two-dimensional grid:

\texttt{examples/data-analysis/Cs2CuBr4_fitting.nb}
As Padé approximants are univariate, the tools require a function to reduce the multivariate high-temperature series to a univariate series for the parameters of a grid point.

The defined function `seriesforparameters` takes the two parameters of a grid point $T_{CW}$, $J_1$ and returns a univariate series in $B = \beta J_1$, where all occurrences of $K$ were substituted by $K = J_2/J_1 \cdot B$. Internally the tools use this function to generate Padé approximants $P[L, M](B)$ for fixed $T_{CW}$.

To derive a function for the susceptibility $\chi(T)$ with the physical units from a Padé approximant another (higher-order) function has to be defined. This function takes the approximant $P[L, M](B)$, the parameters of the grid point $T_{CW}$ and $J_1$, as well as the experimental data and returns a function $\chi(T)$, where the temperature $T$ remains the only undetermined variable.

In our example the function `modelfromapproximant` creates $\chi(T)$ from an approximant by substituting $B$ for fixed $J_1$, multiplying by the constant $4A = 4C/g^2$ and $1/T$, and fitting the remaining linear $g^2$ factor to the experimental data using Mathematica’s `FindFit` function.
Finally we construct Padé approximants and a corresponding model function \( \chi(T) \) for each grid point \( T_{CW,J} \) of the parameter scan and compute for each function the sum of the squared residuals

\[
R = \sum_i (\bar{\chi}_i - \chi(T_i))^2
\]  

against the experimental data points \((T_i, \bar{\chi}_i)\). We named the sums of squared residuals on the grid for a fixed Padé parameter pair \([L, M]\) a residual landscape. For the involved task of computing those residual landscapes for multiple Padé parameter pairs, the supplied Mathematica package offers a convenient function

\[
\text{ComputeResidualLandscapes}[\text{seriesForParametersF}, \text{symbol},
\quad \text{filterPredicates}, \text{modelFromPadeApproximantF}, \text{data}, \text{parameterset}]
\]

The most important parameter is the \text{parameterset} we encountered before, which defines the grid and contains the list of Padé parameters \([L, M]\) for which the residual landscapes are calculated. The function returns a list, with an residual landscape — a two-dimensional array containing the sums of squared residuals for each grid point — for each Padé parameter pair \([L, M]\) of the parameter set. The \text{seriesForParametersF} parameter is the user defined function to generate a univariate series for a grid point and \text{symbol} is the symbol of the variable of this series. These arguments will be used to build Padé approximants. The \text{filterPredicates} parameter takes a list of predicates which are applied to the resulting approximants. Approximants not fulfilling the predicates are immediately dropped.

The next parameter, \text{modelFromPadeApproximantF}, defines how to build a model function \( \chi(T) \) from the approximant. It expects a function taking four arguments, the approximant, the two parameters of a grid point, and the data for the fit. The last parameter \text{data} is the set of experimental data points.

For the example the function is called as follows

\[
\text{residuals12thorder} = \text{ComputeResidualLandscapes}[\text{seriesForParametersF}, B,
\quad \text{filterPredicates}, \text{modelFromPadeApproximantF}, \text{data}, \text{parameterset}];
\]

where \text{HasNoCloseRootPolePair[0.1]} is a predicate to filter all defective Padé approximants, i.e. approximants a root-pole pair in the complex plane with a distance less than 0.1.

Once the computation is completed, \text{residuals12thorder} contains twelve residual landscapes for the Padé parameters \([L, 12 - L], 0 \leq L \leq 12\). From this list of landscapes the best fitting Padé of each grid point can be selected by

\[
\text{minimalresiduals} = \text{SelectMinima}[\text{residuals12thorder}, \text{parameterset}];
\]
returning a list with two elements, where the first element is the minimum sums of
squared residual for each grid point, and the second element is a two-dimensional
array of Padé parameter pairs \([L, M]\) exhibiting the minimum of the corresponding
point.

5.5 Results

For the Heisenberg model on the anisotropic triangular lattice depicted in figure 5.1, we obtained the high-temperature series for the magnetic susceptibility
up to 12^{th} order and the series for the static spin structure factor up to 10^{th} order.
The coefficients of the plain high-temperature series can be found in table B.1 and
the example folder of the program sources, respectively. In the following we com-
pare the computed series to results from Quantum Monte Carlo simulations on the
same lattice for the case of an antiferromagnetic square sub-lattice with ferromag-
netic couplings along the diagonal chains \(J_1 > 0, J_2 < 0\) and we compare the fully
antiferromagnetic case \(J_1 > 0, J_2 > 0\) with experimental results on \(Cs_2CuBr_4\).

5.5.1 Comparison to Quantum Monte Carlo

In contrast to the fully antiferromagnetic case, the mixed case with ferromagnetic
couplings along the chains \(J_1 > 0, J_2 < 0\) is not frustrated and can be treated effi-
ciently with Quantum Monte Carlo methods as it is not subject to the well-known
sign-problem. We performed Quantum Monte Carlo simulations for three exempla-
ry cases \(J_1/J_2 = \{1/2, 1, 2\}\) using the dirloop_sse application of ALPS \[1][86],
which is based on the stochastic series expansion (SSE) method. Each data point
is computed from \(5 \cdot 10^4\) samples on a lattice patch of \(64^2\) sites. Figure 5.2 shows
the temperature behavior of the magnetic susceptibility for \(J_1/J_2 = 1\) obtained
from Quantum Monte Carlo together with the non-defective Padé approximants
\(P[L, M]\) with \(11 \leq L + M \leq 12\) extrapolating the calculated 12^{th} order series.
Above \(T \approx 1.2J_1\) the spread of the Padé approximants is negligible and the
approximants are in excellent agreement with the Monte Carlo results. Below \(T \approx J_1\)
the different Padé approximants start to spread drastically. However, even in this
temperature regime, down to \(T \approx 0.6J_1\), a few approximants are found to agree
well with Monte Carlo. In this regime, however, a prediction solely based on the
computed series would be hard to justify, due to the broad spread of the dif-
f erent approximants. For the other parameters \(J_1/J_2 = \{1/2, 2\}\), a very similar
spreading behavior and agreement with Quantum Monte Carlo data of the Padé
approximants is observed (not shown).

Also for the spin structure factor \(S(k)\) (fig. 5.3), the 10^{th} order series and
Quantum Monte Carlo estimates agree beautifully down to \(T \approx 1.8J_1\) or rather
5.5 Results

Figure 5.2: Comparison of Padé approximants $P[L,M]$ with $11 \leq (L+M) \leq 12$ to the $12^{th}$ order series of the magnetic susceptibility $\chi(T)$ for a non-frustrated system with ferromagnetic chain couplings ($J_1 = 1, J_2 = -1$) with Quantum Monte Carlo simulations.

$T \approx 1.4J_1$, depending on the particular couplings. For lower temperatures the predictions from the series data become increasingly unreliable as the Padé approximants start spreading. Both methods consistently display the peak at $M'$, which is the residue of antiferromagnetic ordering at $T = 0$.

5.5.2 Comparison with Cs$_2$CuBr$_4$

For Cs$_2$CuBr$_4$ we fitted the 12$^{th}$ order series to the experimental data [87] on the magnetic susceptibility. The model functions $\chi(T)$ with proper units are obtained from the bare series as described in the previous section. Figure 5.4 shows the sum of squared residuals compared to the experimental data of the best fitting non-defective Padé approximants $P[L,M]$ with $9 \leq L + M \leq 12$ for each point of the grid. Note that we only consider $T > 7K$ for the procedure, as the approximants will not approximate the underlying function well for low temperatures. The cutoff temperature was chosen slightly below the point the different approximants start to spread significantly. Higher order calculations might reduce this spreading temperature further, however, we do not expect significant changes with 1-2 additional orders, as we see only marginal improvement from 10$^{th}$ to 12$^{th}$ order.
Figure 5.3: Comparison of the static structure factor $S(k)$ computed by series expansions (lines) against Quantum Monte Carlo simulations (points with error-bars) for a non-frustrated systems with ferromagnetic chain couplings $J_1 > 0$, $J_2 < 0$ at $T = 1.8J_1$. The path through the Brillouin zone is shown on the left. While discrete error-bars represent the standard deviation of Monte Carlo samples, the continuous error-areas around the lines represent the spread of Padé approximants and are not rigorous error estimates.

The parameter scan reveals two shallow elliptical valleys of the residuals centered around the local minima at $T_{CW} = 12.5K$, $J_1 = 4.9K$ and $T_{CW} = 12.8K$, $J_1 = 11.6K$, respectively. For both points the corresponding Padés behave remarkably similar. Both Padés yield an excellent fit to the experimental data within the considered temperature range (fig. 5.5). Also both Padés slightly overestimate the susceptibility for high-temperatures and slightly underestimate close to the maximum, before both Padés diverge around $T = 6K$.

However, also parameters at the verge of the valleys of the residual landscape, e.g. at $T_{CW} = 11.3K$, $J_1 = 3.5K$ and $T_{CW} = 14.2K$, $J_1 = 11.8K$ ( (c) and (d) in figures 5.4 and 5.5), fit the data very well and illustrate how a wide range of parameters along the valleys lead to comparable fits. The extracted $g$-factors of all presented approximants lie within the range expected from the Curie-Weiss law, but underestimate the $g$-factor obtained from spin resonance experiments $g = 2.09$ [84].
Physically, the valleys correspond to two different regimes. In case (a) the triangular model is dominated by the antiferromagnetic chain couplings \( J_2 = 15.2 \)K competing with a significantly smaller square lattice coupling of only \( J_1 = 4.9 \)K \( (J_1/J_2 = 0.3(2)) \). This is also reflected in static structure factor \( S(k) \) shown in figure 5.6, which features a plane-wave structure along the axis of the chain couplings, with maxima close to the antiferromagnetic ordering vector of decoupled chains. The plane wave is only slightly wiggled by the competing square lattice couplings, pushing the maxima further to the corners of the Brillouin zone to \( Q = 0.54b_j^2 \), where \( b_j \) is the reciprocal vector corresponding to the lattice vector \( a_j \) along the \( J_2 \) bonds. The other valley (b) corresponds to a triangular lattice with a strong square lattice coupling \( J_1 = 11.6 \)K perturbed by only a very small chain coupling \( J_2 = 2.4 \)K \( (J_2/J_1 = 0.2(1)) \). Here the structure factor exhibits a peak at the symmetry point \( M' \) as one would expect from an antiferromagnetic ordering on the square lattice.

Previous studies of \( \text{Cs}_2\text{CuBr}_4 \) using high-temperature series expansions by...
Zheng et al. [85] obtained $J_1 = 6.99K$ and $J_2 = 14K$ ($T_{CW} = 14K$, $J_1/J_2 = 0.5$) from the $10^{th}$ order series of the susceptibility, but also report that a wide range of ratios $0.35 < J_1/J_2 < 0.55$ give comparable fits. Those values reside at the upper end of our residual valley (a). Another study extracted the Heisenberg couplings from high-field electron-spin-resonance measurements with harmonic spin-wave theory [84]. Their findings of $J_1 = 6.1(3)K$ and $J_2 = 14.9(7)K$ corresponds to a Curie-Weiss temperature of $T_{CW} = 13.6(2)K$ and $J_1/J_2 = 0.41(0)$. While also their coupling ratio seems significantly higher than the $J_1/J_2 = 0.3(2)$ obtained here, the parameters lie well within valley (a) (fig. 5.4) and are plausible candidates.

Further evidence for case (a) are neutron-scattering experiments on $Cs_2CuBr_4$ [88, 89], which revealed an incommensurate ordering vector of $Q = 0.575b_J$ at zero magnetic field, consistent with our predictions from the high-temperature series for point (a) (fig. 5.6). While the predicted ordering vector for fit (a) $Q = 0.54b_J$ is slightly smaller than the ordering vector measured by neutron-scattering, one has to bear in mind that they were obtained at significantly different temperatures of $T = 15K$ and $T = 60mK$, respectively, and might further converge. Zheng et al. [85] and Ono et al. [89] also estimate the coupling ratio $J_1/J_2$ from the ordering vector by comparison with various theoretical models: for the ground-state of the classical spin model on an anisotropic triangular lattice, where neighboring
5.6 Extensibility

The presented applications for Heisenberg models are built with a modular C++ framework for high-temperature series expansions we developed. The framework
provides generic algorithms for the graph generation, the high temperature expansion, and the graph embeddings. With little effort the presented application could be generalized to include Dzyaloshinskii-Moriya terms or new applications for fermionic lattice models, such as the $t$-$J$ can be written based on the existing implementation of the algorithms.

5.7 Conclusion

We presented a set of applications and Mathematica packages to compute and analyze high-temperature series for the free energy, the uniform magnetic susceptibility and the static structure factor of Heisenberg models on arbitrary lattices with up to four independent coupling constants. As an example we computed the high-temperature series for the anisotropic triangular lattice and showed how to obtain estimates for the coupling constants by fitting the Padé approximants to susceptibility data for Cs$_2$CuBr$_4$.

While the obtained high-temperature series for the anisotropic triangular lattice can be directly applied to other realizations of the same model, such as Cs$_2$CuCl$_4$ [91] or various organic superconducting materials [92, 85], the computer programs allow easy access to high-temperature series of all quantum magnets, in arbitrary dimensions, which can be described by spin-1/2 Heisenberg models on regular lattices, including layered quasi two-dimensional lattices or dimerized systems. Therefore the presented applications may prove to be a powerful tool in the lab, to gain a better understanding experimental results.
5.7 Conclusion
Chapter 6

T=0 Series for the Bose-Hubbard Model

6.1 Introduction

Strongly correlated bosons have received a lot of attention in condensed matter physics within the last 30 years. In general, bosonic systems offer a variety of unique phenomena, such as the Bose-Einstein condensate (BEC) or superfluidity and are essential for the understanding of superconductivity, where two electrons form bosonic Cooper-pairs.

The Bose-Hubbard model is the most basic model to describe bosons on a regular lattice and is one of the exemplary toy models to investigate quantum phase transitions [79]. It describes spinless bosons on a lattice, which can hop to nearest neighbor sites and only interact with the bosons encountered on the same site. The first in-depth study of the model by Fisher et al. [93] predicted three phases at zero temperature: a superfluid phase, an incompressible Mott-insulating phase and, by including a random local potential term, a Bose glass phase. They also laid out the qualitative understanding of the associated Mott-insulator-to-superfluid (MI-SF) quantum phase transition. Since then, the model has been thoroughly examined including various extensions, such as disorder [94, 95], nearest-neighbor repulsion [96, 97] or external potentials [98, 99] with most of the arsenal available to condensed matter physicists, ranging from numerical methods including Quantum Monte Carlo [96, 97], DMRG [100], DMFT [101], to various analytical approaches [102, 103, 104].

In the early 2000s, renewed interest in the model was sparked by the rise of ultra-cold atom experiments in optical lattices [105, 106]. By trapping single atoms in the local minima of a strong regular potential formed by superimposed laser-beams, experimentalists managed to create well-controllable realizations of
6.1 Introduction

Figure 6.1: Lattices geometries for the Bose-Hubbard model. top: triangular, square and honeycomb lattice; bottom: stacked-triangular, stacked-square and stacked-honeycomb lattice. Solid lines represent the intra-layer hopping \( t \), dotted lines the inter-layer hopping \( t' \).

the theoretical lattice models in condensed matter physics. Shortly after the first optical lattice experiments, the predicted MI-SF transition of the Bose-Hubbard model was observed by Greiner et al. for \(^{87}\text{Rb}\)-atoms in a three-dimensional optical lattice [107]. After these pioneering experiments, the transition was also observed in two-dimensional optical lattices, such as the square [108] and the triangular lattice [109]. Direct quantitative comparison of optical lattice experiments and numerical Quantum Monte Carlo simulations revealed remarkable quantitative agreement [98], confirming that optical lattices are simulators for condensed matter systems.

Recently, a Quantum Monte Carlo study of the Bose-Hubbard model on the honeycomb lattice with a staggered sublattice potential suggested the existence of bosonic edge states [99] as known from fermionic systems, where they appear in topological insulators [110] or the honeycomb structured graphene [111]. While such features are not directly accessible with perturbative series expansions, the results motivate a closer look at the physics of bosons on honeycomb lattices. Even though many studies have considered the various extended Bose-Hubbard models on the honeycomb lattice before (e.g. [96, 97, 112, 113]), results for the plain Bose-Hubbard model are surprisingly sparse.

Here we show how to use perturbative series expansions at \( T = 0 \) to obtain the phase diagram of the ground state and investigate the Mott-insulator-superfluid transition for various lattice geometries as depicted in figure 6.1. We obtained 12\(^{th}\) order series for boundary of the \( n = 1 \) Mott phase of the two-dimensional lattices,
namely the square, triangular and honeycomb lattice, and 10th order series for their three-dimensional extensions, the stacked-square, stacked-triangular and stacked-honeycomb lattice. For the \( n = 2 \) Mott state, we computed the 9th and 8th order series, respectively. While for the square and triangular lattice, we merely verified our simulation code by comparing our results to prior calculations by Elstner and Monien [37, 114], we present original new quantitative predictions for the phase diagram and the critical behavior of the closing Mott gap for the honeycomb lattice and the stacked lattices. For the stacked lattices, we derived a perturbation series in separate intra- and inter-layer coupling variables, allowing us to continuously tune the system from a purely 2D lattice to a 3D lattice and observe the change in universality class of the quantum phase transition.

### 6.2 Bose-Hubbard Model

The Bose-Hubbard model describes the interaction of bosonic particles living on a lattice. Its Hamiltonian is given by

\[
H = -t \sum_{\langle i,j \rangle} (b_i^\dagger b_j + h.c.) + \frac{U}{2} \sum_i n_i(n_i - 1) - \mu \sum_i n_i,
\]  

(6.1)

where \( b_i^\dagger \), \( b_i \) are bosonic operators, creating or annihilating a particle on site \( i \), respectively, and \( n_i = b_i^\dagger b_i \) the particle number operator. The first term controls the hopping \( t \) of the bosons along the bonds \( \langle i,j \rangle \), the second term is the on-site interaction \( U \) of the bosons, which is repulsive for \( U > 0 \) or attractive for \( U < 0 \). The chemical potential \( \mu \) controls the total number of particles in the system. When applied to optical lattices, the chemical potential \( \mu \) is usually site-dependent due to the confinement potential of the optical trap, but here we only consider a uniform global chemical potential.

The competition between the local on-site repulsion \( U \) and the hopping \( t \) leads to distinct ground state phases as shown in the schematic phase diagram in figure 6.2. For small \( t/U \), the system exhibits gapped Mott-insulating phases, the so-called Mott lobes, with a fixed commensurate density of \( n \) bosons per lattice site. In this phase the bosons are localized, since the excitation of a particle-hole pair would lead to a higher occupied site at the cost of additional repulsion energy and is therefore gapped. For larger \( t/U \), the model features a superfluid phase where the bosons coherently delocalize over the full lattice.

In the following, we will compute the boundaries of the Mott lobes and investigate the critical behavior at the tip of the Mott lobe. Close to the quantum phase transition at the tip, the energy gap \( \Delta \) of the Mott state closes and is expected to show the critical power law behavior

\[
\Delta(t) \propto |t - t_c|^{\nu}, \quad |t - t_c| \ll 1,
\]  

(6.2)
6.2 Bose-Hubbard Model

where \( z = 1 \) is the dynamical critical exponent and \( \nu \) is the critical exponent to be determined.

We study the model using perturbative series expansions at \( T = 0 \), where we perturb the system from the atomic limit of isolated sites, each hosting \( n \) bosons, by ramping up the hopping term. We therefore drive the system, initially in a commensurate Mott state, towards the phase transition to the superfluid phase at \( t_c \), while retaining the commensurate density. The Hamiltonian, split into an unperturbed system \( H_0 \) and a perturbation, reads

\[
H/U = H_0 + xV = \sum_i n_i(n_i - 1) - \frac{t}{U} \sum_{\langle i,j \rangle} (b_i^\dagger b_j + h.c.),
\]

(6.3)

where the hopping-repulsion ratio \( x = t/U \) serves as perturbation parameter.

For the ground state energy series \( E_0(x) \) of the Mott state, the procedure starts from the unperturbed ground state of the atomic limit, where each site is occupied by \( n \) bosons

\[
|n\rangle = \prod_{i=0}^{N} \frac{1}{\sqrt{n!}} (b_i^\dagger)^n |0\rangle,
\]

(6.4)
with a ground state energy per site of
\[ E_0(x = 0)/L = \frac{1}{2} n(n - 1) - n\mu/U. \]  (6.5)

The Mott lobes are bound by the energy of adding and removing a particle from the system, i.e.
\[ E^p = +n - \mu/U \quad \text{(particle)}, \]  (6.6)
\[ E^h = -(n - 1) + \mu/U \quad \text{(hole)}. \]  (6.7)

When switching on the hopping, \( x \neq 0 \), the additional particle or the created hole start to delocalize and the system can be described by the perturbed effective Hamiltonians \( H_{\text{eff}}^p \) and \( H_{\text{eff}}^h \), respectively. These Hamiltonians are calculated as matrix elements of the unperturbed single-particle states
\[ |p, i; n\rangle = \frac{1}{\sqrt{n + 1}} b_i^\dagger |n\rangle, \]  (6.8)
or single-hole states
\[ |h, i; n\rangle = \frac{1}{\sqrt{n}} b_i |n\rangle, \]  (6.9)
where one boson is added to (removed from) the Mott state \( |n\rangle \) on site \( i \). Due to translational invariance of the lattice, the resulting Hamiltonians can be trivially diagonalized by Fourier-transformation to \( k \)-space. For \( x \neq 0 \), the energies \( E^p, E^h \) therefore become
\[ E^p(x) = +\omega_0^p(x) - \mu/U, \]  (6.10)
\[ E^h(x) = -\omega_0^h(x) + \mu/U, \]  (6.11)
where \( \omega_0^{p/h} \) are the minimal eigen-energies of the effective Hamiltonian for the particle and hole states, respectively.

The main computational challenge for perturbative series expansions of the Bose-Hubbard model is the faster than exponentially growing Hilbert space. For the Bose-Hubbard model all particles of a given graph could potentially occupy a single site, therefore adding a site to a \( n \) Mott state, will not only increase the dimension of the global Hilbert space by a fixed additional factor, but also increase the local Hilbert space of all other sites in the graph. Given that the number of bosons is conserved, the dimension of the Hilbert space for a particle on the \( n \) Mott state is given by distributing \( n \cdot L + 1 \) bosons on the \( L \) sites of the graph
\[ \dim(\mathcal{H}_g) = \binom{L + L \cdot n}{L \cdot n + 1} = \frac{(L + L \cdot n)!}{(L - 1)!((L \cdot n + 1)!}. \]  (6.12)

For the largest contributing graph of the 12\(^{th}\) order series, having 13 sites, this amounts to 9'657'700 states. However, at low orders of the perturbation not all of those states appear in the calculation and a sparse implementation is advisable.
6.3 Running the Application

From a user perspective, running the application is very similar to the high-temperature series expansions for the anisotropic Heisenberg model discussed in section 5.3. However, an additional step reduces the number of graphs and the weight computation is split into three independent runs for the ground-state energy of the Mott state and the effective Hamiltonians of the first excited sector: the single-particle and single-hole excitations. The full recipe to obtain the 8th order series for the particle-hole excitation gap of the $n = 1$ Mott phase comprises seven steps:

1. the graph generation, creating a list of all embeddable graphs for a given lattice. This step is identical to the high-temperature expansion and is described in more detail in section 5.3.1

```bash
generate_graphs latticeparam.json graphlist.graphs
```

2. filtering the graphs to exclude elements that do not contribute to a perturbative series expansion

```bash
filter_graphs --n-edge-paths 10 graphlist.graphs
⇒ contributing.graphs
```

3. the weight computation for the ground-state energy, calculating the contribution of each graph to the ground-state energy series

```bash
bose_hubbard_gse_compute_weights_1_variables 1 0 8
⇒ contributing.graphs groundstate.series
```

4. the weight computation for the effective Hamiltonian of the single-particle sector

```bash
bose_hubbard_gse_compute_weights_1_variables 1 1 8
⇒ contributing.graphs Heff_particle.series
```

5. the embedding step for the particle sector, which first subtracts the ground state contribution from the diagonal of the effective Hamiltonian for each graph, followed by the subcluster-subtraction of the weights and embedding the irreducible weights into the lattice

```bash
bose_hubbard_gse_reduce_embed_weights latticeparam.json
⇒ graphlist.graphs contributing.graphs groundstate.series
⇒ Heff_particle.series
```

At this point the full effective Hamiltonian for the single-particle sector is obtained. For the effective Hamiltonian of the single-hole sector, the last two steps have to be repeated with slightly different parameters:
Figure 6.3: Exemplary interaction diagrams of a $H^n$-term depicted as continuous path on the graph. Left: an interaction described by a $H^5$-term on a non-contributing graph with 5 edges and the same interaction diagram on a smaller graph. Right: a contributing $H^6$-interaction on the same graph.

6. the weight computation for the effective Hamiltonian of the single-hole sector

   \begin{verbatim}
   $\text{bose_hubbard\_gse\_compute\_weights\_1\_variables}$ 1 -1 8
   $\rightarrow$ contributing.graphs Heff\_particle.series
   \end{verbatim}

7. and the embedding step for the hole sector

   \begin{verbatim}
   $\text{bose_hubbard\_gse\_reduce\_embed\_weights}$ latticeparam.json
   $\rightarrow$ graphlist.graphs contributing.graphs groundstate.series
   $\rightarrow$ Heff\_hole.series
   \end{verbatim}

### 6.3.1 Graph Generation and Filtering

For an $N^{th}$ order perturbation series at $T = 0$, the initial step of generating all graphs with $N$ edges for the lattice under investigation is identical to the high-temperature expansion and described in section 5.3.1. However, in contrast to the high-temperature expansion of the correlator many of those graphs do not contribute even for the calculation of the single-particle dispersion. For a graph to have a non-zero irreducible weight for the single-particle effective Hamiltonian in $N^{th}$ order, the graph has to provide a contiguous path of exactly $N$ hops visiting all edges at least once. This can be understood by imagining the superpositions of diagrams. Figure 6.3 shows a graph with $N = 5$ edges. However, there is no $5^{th}$ order interaction diagram which is not covered by a graph with less edges. Therefore, all interaction diagrams in the bigger graph will cancel in the subcluster-subtraction and the graph will not contribute in $5^{th}$ order.

For filtering out graphs that fulfill predefined criteria, such as hosting a contiguous path covering all edges, we provide the tool

   \begin{verbatim}
   filter\_graphs <filter\_flag> <filter\_args> <inputfile> <outputfile>
   \end{verbatim}
which reads all graphs of the \texttt{inputfile}, applies the filter and stores the graphs meeting the filter requirements in the \texttt{outputfile}.

The arguments \texttt{filter\_flag} and \texttt{filter\_args} select the filter type and provide filter-defined parameters to the filter. The tool provides two different filter types:

- \texttt{--less-than-n-edges} selects a simple edge count filter, which copies all graphs with \( n \leq N \) edges to the output file. It requires the maximal number of edges \( N \) as \texttt{filter\_args}. This filter is helpful if graphs for a high-order calculation for a lattice were generated before and a quicker lower-order computation for the lattice is desired. Instead of generating the graphs from scratch for the lower order — which may be time consuming for larger number of edges — the tool may simply extract the graphs required for the calculation.

- \texttt{--n-edge-paths} selects the path filter required here. This filter tries to find a path with \( N \) hops covering all edges of the graph. If such a path is found the graph is copied to the output file. The \texttt{filter\_args} for this filter is the number of \( N \) hops of the path.

Discarding all graphs which cannot host such a path is essential for high-order expansions as it reduces the number of graphs to be considered in the weight computation drastically. Especially the most of the computationally demanding graphs, namely the graphs with exactly \( N \) edges, are dropped, since only the linear graph with \( N \) edges and \( N + 1 \) vertices features such a path.

For the 12\textsuperscript{th} order expansion on the triangular lattice, this step saves almost 85\% of the required calculations: while there exist 28'367 embeddable graphs with at most 12 edges for the triangular lattice, only 4'604 can host the required path. For higher orders the numbers become even more extreme. For example for a 14\textsuperscript{th} order expansion only 33'523 of the 339'028 embeddable graphs may contribute.

6.3.2 Weight Computation

The weight computation is the main difference between the high-temperature and the \( T = 0 \) perturbation series expansions. Instead of calculating the partition function \( Z \) and related quantities as weights, ground-state energy and effective Hamiltonians are computed by Rayleigh-Schrödinger perturbation theory. However, even though the computations are conceptually different, the weight computation steps are the computationally most demanding part of the two expansion methods for the same reason: the exponentially growing Hilbert space with the number of vertices of the graphs. For the users of the computer programs the differences are nevertheless mostly hidden. Since the high-temperature expansions and the perturbative \( T = 0 \) expansion programs are based on the same C++
framework, many aspects of this step will actually remind the reader of the weight computation step in the previous chapter. Like for the Heisenberg model we provide computer applications for up to four independent hopping parameters

1. `bose_hubbard_gse_compute_weights_1_variables`
2. `bose_hubbard_gse_compute_weights_2_variables`
3. `bose_hubbard_gse_compute_weights_3_variables`
4. `bose_hubbard_gse_compute_weights_4_variables`

Independent of the number of variables, the applications take the arguments

```
   ... <mott_n> <mott_charge> <order> <graphfile> <outputfile>
```

where

- `<mott_n>` sets the (integer) number of bosons per site of the Mott state to be calculated.
- `<mott_charge>` switches between calculations for
  
  0: the ground-state energy $E_0$
  
  1: the effective Hamiltonian $H_{\text{eff}}^{p}$ of an additional boson on the Mott state.
  
  -1: the effective Hamiltonian $H_{\text{eff}}^{h}$ of a hole in the Mott state.
- `<order>` sets the order of the perturbation expansion
- `<graphfile>` is the name of the file containing the list of graphs for which the series should be calculated, and
- `<outputfile>` is the name of the output file where the weights will be stored.

Just like the high-temperature expansion applications, the programs will not overwrite any results in the output file, but only calculate and append weights of graphs not yet found in the file. This allows to continue the computation after an interruption by simply re-executing the command, as well as to extend the calculations for new lattices later, reusing the already computed weights.

For high-order expansions on clusters or supercomputers we again provide MPI-parallelized versions of the programs, which take two additional optional parameters compared to the standard versions:

```
   ... <mott_n> <mott_charge> <order> <graph_file> <series_file>
   ↦ [checkp_interval] [task_size]
```
6.3 Running the Application

where the parameters `checkp_interval` and `task_size` are already known from
the high-temperature expansion programs for the Heisenberg model. In contrast to
the high-temperature expansion, the perturbation expansions do not use threading
within an MPI process. Thus, for optimal usage of the computational resources,
the program should be run with as many MPI processes as CPU-cores, rather than
compute nodes. In many cases, however, this optimal scenario is not possible due
to the rapidly growing memory requirements with the graph size. In this case
fewer MPI processes should be distributed on the compute nodes evenly, such that
a few processes per node may reserve the full memory of the node.

For computing the energy gap $\Delta(t/U)$ of the Bose-Hubbard model, the program
has to be executed at least three times as shown in the beginning of this section:
one for the ground-state energy series of the Mott state and once for each of the
effective Hamiltonians $H^p_{\text{eff}}, H^h_{\text{eff}}$ describing the physics of the particle and the hole
states.

### 6.3.3 Subcluster-Subtraction and Embedding

The last step, the subcluster-subtraction and the embedding of the graphs with
their associated weights into the lattice, is done by the program

```
bose_hubbard_gse_reduce_embed_weights <parameterfile> <full_graphfile>
  ↦ <contributing_graphfile> <gs_seriesdb> <Heff_seriesdb>
```

where the parameters are

- `<parameterfile>`, the name of the lattice parameter JSON file,
- `<full_graphfile>`, the name of the file containing *all embeddable* graphs for
  the lattice with up to $N$ edges,
- `<contributing_graphfile>`, the name of the file containing only the graphs
  having (potentially) non-zero irreducible weights, i.e. the filtered graphs of
  the first step,
- `<gs_seriesdb>`, the filename of the ground-state series weights file and
- `<Heff_seriesdb>`, the filename of the effective Hamiltonian weights file.

The program will first subtract the ground-state energy series from the diagonal
of the effective Hamiltonians, such that the matrices become cluster-additive and
thus suitable for the linked-cluster expansion (see section 2.3). After the following
subcluster-subtraction and embedding of the ground-state energy weights and the
effective Hamiltonian, the program will print out the full ground-state energy series
for the lattice as well as the effective Hamiltonian Fourier-transformed to $k$-space.
Due to translational invariance the Fourier-transformation will diagonalize the Hamiltonian for lattices with a single site in the unit cell. For lattices with \( m \) sites per unit cell, such as the honeycomb lattice, the printed Hamiltonian will be a continuum of \( m \times m \) matrices \( M(\mathbf{k}) \) describing the sublattice interactions.

### 6.4 Series Analysis

Further analysis of the ground state energy series and the perturbative effective Hamiltonian is done using the Mathematica package already known from the analysis of the high-temperature series for the Heisenberg model\(^1\). While the analysis is also possible without the package, it provides functions for common procedures which make the analysis considerably more convenient. The full analysis for the stacked-triangular and the honeycomb model can be found in the Mathematica notebooks\(^2\) shipped as examples with the code. Note that the notebook for the stacked lattices has one additional layer of complexity, compared to the example presented here, as each step is done for a list of intra-inter-layer hopping ratios \( t'/t \) by mapping the operations to the list.

In the first step, we load the Mathematica module and the obtained series the effective Hamiltonians \texttt{niparticle} and \texttt{nihole} from text files containing the output of the subcluster-subtraction and embedding step.

```mathematica
<< "padeanalysis.m"
niparticle = << "particleseries_10edges.txt"
nihole = << "holeseries_10edges.txt"
```

The series are symbolic in the hopping amplitudes \( X = t/U \), \( K = t'/U \) and the \( \mathbf{k} \)-vector components \((k_0,k_1,k_2)\). As we calculated the series for 12\(^{th}\) order for the individual graphs, while we only considered embeddable graphs with up to 10 edges for this particular lattice, the series are only valid up to 10\(^{th}\) order. We trim the series to include only terms \( X^nK^m \) up to order \( n + m \leq 10 \):

```mathematica
niparticle = TrimSeries[niparticle, 10, \{X, K\}];
nihole = TrimSeries[nihole, 10, \{X, K\}];
```

Those series already represent the dispersions \( \omega^{b/h}(t/U,t'/U; \mathbf{k}) \) and may be plotted for any hopping amplitudes \( t, t' \ll 1 \) using Mathematica’s standard plotting functions. In order to extrapolate the series to larger values of \( t \) and \( t' \) and to gain qualitative error estimates, we process the series further by forming Padé approximants.

---

\(^1\)padeanalysis.m in /opt/lcse/share/lcse/data_analysis after installation.
\(^2\)Bose_Hubbard_stacked-tri.nb, Bose_Hubbard_honeycomb.nb in examples/data_analysis
6.4 Series Analysis

6.4.1 Dispersion Functions

From here on, we only describe the procedure for the particle Hamiltonian $H_{\text{eff}}^p(k)$. The hole Hamiltonian $H_{\text{eff}}^h(k)$ is processed in the same manner.

For the Padé approximation we assume an Hermitian effective Hamiltonian, such that the dispersion function $\omega(k) = n_{\text{particle}}$ is a real-valued function which can be decomposed into a sum of cosine terms. We split the function into a list of pairs, where the first element is the argument of the cosine and the second element the corresponding amplitude, and call the resulting list $k$-space term table. By construction the amplitudes are series of the perturbation parameters, whereas the cosine arguments only involve the $k$-vector. We provide the BuildKSpaceTermsTable function which performs simplification of the $n_{\text{particle}}$ dispersion, brings it to the desired cosine form and creates the table

```mathematica
kstt = BuildKSpaceTermsTable[nparticle];
```

For each element of the $k$-space term table, we fix the all but one hopping terms (here: $K\rightarrow1/2*X$) and form the Padé approximants of the amplitude part for a given set of orders

```mathematica
orders = {8,9,10};
kstpadelist = Map[
  Simplify[
    {#[[1]], PadeApproximants[#[[2]] /. K -> 1/2*X, X, orders]}
  ] &, kstt];
```

The resulting table is a list of pairs, where the first element remains the cosine argument, and the second element is a list of Padé approximants of the respective amplitude.

From the list Padé approximants for each term, we filter the non-defective ones by applying the FilterPades function to each Padé list of the table.

```mathematica
kstpadesfiltered = Map[
  {#[[1]],
    FilterPades[ HasNoPoleWithSmallerResidueCloserThan[0.003, 0.04],
      #[[2]], X ] }
  ] &, kstpadelist];
```

The FilterPades takes a predicate or a list of predicates, the list of Padé approximants to be filtered and the free variable in the approximant. It returns a new list which contains only the approximants for which all predicates were true. In this example we accept only Padé approximants without close root-pole pairs in
vicinity of the origin. More precisely, we only accept approximants $P(x)$, which have no pole $x_0$ with a residue $\text{Res}_{x_0}(P(x)) \leq 0.003$ within the expected “physical” disc $|x| < 0.04$ in the complex plane. The size of the physical disc is given by a guess of the critical hopping $t_c$, where we expect our perturbation series to become invalid, and the cutoff value for the residue is an empirical value which was initially suggested by Hunter and Baker [115]. We can get the number of remaining approximants by

$$\text{ListNumberOfPadesForEachTerm}[\text{kstpadesfiltered}, \text{orders}]$$

which returns a list of cosine terms with the number of approximants for its amplitude.

The $k$-space term table is then recomposed to a sum of cosine terms again, by constructing the functions $\text{dispmean}[x]$ and $\text{disperror}[x]$,

$$\text{dispmean} = \text{MeanKSpaceApproximants}[\text{kstpadesfiltered}, X, \text{orders}]$$
$$\text{disperror} = \text{ErrorKSpaceApproximants}[\text{kstpadesfiltered}, X, \text{orders}, \text{ErrorFunction} \rightarrow \text{MaxAbsDeviation}]$$

where, for a given value $x = t/U$, the former evaluates the mean for each amplitude from the list of non-defective approximants and the latter the error of the amplitudes, based on the maximum absolute deviation.

Finally, we evaluate these functions along a path in the Brillouin zone

$$\text{path} = \{\{0, 0, 0\}, \{0, 2\pi/\sqrt{3}, 0\}, \{2\pi/3, 2\pi/\sqrt{3}, 0\}, \ldots\}$$
$$\text{meandata} = \text{ComputeValuesAlongPath}[\text{dispmean}[0.026], \{k0, k1, k2\}, \text{path}]$$
$$\text{errordata} = \text{ComputeValuesAlongPath}[\text{disperror}[0.026], \{k0, k1, k2\}, \text{path}]$$

where $\text{meandata}$ and $\text{errordata}$ are tables of a continuous index $i$, the components of $k$ and the function value $f(i, k0, k1, k2, f)$. The data can be plotted using the first and fifth column

$$\text{ListLinePlot}[\text{Map}[\{\#[[1]], \#[[5]]\} &, \text{dispdata}]]$$

### 6.4.2 Gap Series and Critical Properties

From the particle and hole dispersions we find the energy gap $\Delta(x) = \omega^p + \omega^h$ for exciting a particle-hole pair on the Mott state at $k = 0$. We obtain the series for the gap by substituting $k$ accordingly

$$\text{ngap} = \text{Simplify}[\text{(n1particle + n1hole) /. \{k0 -> 0, k1 -> 0, k2 -> 0\}}]$$

From a simple plot of the gap over $X = t/U$ for fixed $K/X$, we get a rough guess for the physical disc radius $\text{nphysicaldisc}$ to filter the Padé approximants. We choose the root of the gap series and add a 15% margin. The particular value is not important, since we manually select good approximants in a second step. However,
6.4 Series Analysis

choosing a too large value may discard potentially good approximants, while a too small value filters less defective approximants automatically and increases the manual work.

We again fix all but one hopping variable \( X \) and create a table of DLog-Padé approximants of 7th, 8th and 9th order

```mathematica
n1gapdp = DLogPadeApproximants[n1gap /. \{K -> 1/2*X\}, X, \{7,8,9\}];
```

Note that for a 10th order series the highest DLog-Padé approximant is of 9th order, since we “lose” one order in the derivative. As before, we filter the approximants

```mathematica
n1gapdpfiltered = FilterPades[{HasNoPoleWithSmallerResidueCloserThan[0.003, n1physicaldisc]}, n1gapdp, X];
```

and drop any approximant with a small residue within our estimated physical disc radius \( n1physicaldisc \).

For each of the remaining approximants we compute a list of all poles and their residues in the complex plane

```mathematica
n1PolesAndResfiltered = GetPolesWithResidues[n1gapdpfiltered, X, WorkingPrecision -> 100];
PrintPolesWithResiduesTable[n1PolesAndResfiltered]
```

and print it in a convenient table. We used a high working precision in this step, since the large numbers in the DLog-Padé approximants may lead to residues being evaluated to 0 erroneously. The poles returned by \( \text{GetPolesWithResidues} \) are sorted by their distance to the origin. In the table we carefully examine the poles for close root-pole pairs and confluent singularities close to the critical hopping \( t_c \), and create a list of the parameters of the non-defective approximants

```mathematica
n1gapgoodpades = {{0, 9, Null}, {8, 1, Null}, ... };
```

We select the pole and residue data of those approximants and create a list of the first pole, i.e. the pole closest to the origin, of each approximant and a list of their residues

```mathematica
polereslist = GetPadeDataList@SelectPadesByParameters[n1gapgoodpades, n1PolesAndRes];
polelist = Map[#[[1,1]] &, polereslist];
residuelist = Map[#[[1,2]] &, polereslist];
```

Assuming that the closest pole of each DLog-Padé approximant is the critical value of interest, we obtain the estimates for the critical hopping \( t_c \) and the critical exponent \( z \cdot \nu \) of the gap \( \Delta(t/U) \) from the mean value of the lists and estimates for the errors by the maximum absolute deviation

```mathematica
Mean[polelist]  
MaxAbsDeviation[polelist]  
Mean[residuelist]  
MaxAbsDeviation[residuelist]
```
6.5 Results

6.5.1 2D Lattices

We obtained the perturbation series in $x = t/U$ for the ground state energy $E_0(x)$ of the $n = 1$ and $n = 2$ Mott states up to $12^{th}$ and $9^{th}$ order, respectively, for the triangular, square and honeycomb lattices. Up to the same order, we computed the effective single-particle and single-hole Hamiltonians $H_{eff}^p$ and $H_{eff}^h$ on those Mott states.

The effective Hamiltonians were diagonalized by Fourier-transformation and, in case of the honeycomb lattice, diagonalization of the 2x2 matrix describing the interaction of the two sublattices of the bipartite lattice. For all lattices the resulting dispersions of the dressed single particles $\omega^p(k)$, as well as single holes $\omega^h(k)$, develop minima at the Γ-point for $t \neq 0$. The graphs of the dispersions calculated from the $12^{th}$ order series are indistinguishable from the first $[\Gamma, \ldots, \Gamma]$ segments of the $t'/t = 0$ dispersion lines of the $10^{th}$ order data for the stacked versions of the lattices presented in the next section. Therefore we do not show separate figures for the two dimensional lattices, but refer the reader to the figures figs. 6.4–6.6 of the respective stacked lattices. Note that the figures show the negative hole dispersion $-\omega^h(k)$ to illustrate the energy gap $\Delta(x) = E^p(x) - E^h(x)$ of particle-hole pair.

The minima at the Γ point ($k = 0$) define the boundaries of the Mott phase $\omega^p/h_0 = \omega^p/h(0)$. Also for the phase diagram we refer the reader to the respective figures and discussion of the stacked-lattices in the next section (figs. 6.8–6.10), as the two additional orders of the series here have no significant effect on the computed phase boundaries.

<table>
<thead>
<tr>
<th>Lattice</th>
<th>$t_c/U$</th>
<th>$\nu$</th>
<th>$N_{\text{Padés}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>triangular</td>
<td>$0.03779 \pm 0.00003$</td>
<td>$0.684 \pm 0.005$</td>
<td>12</td>
</tr>
<tr>
<td>square</td>
<td>$0.05976 \pm 0.00023$</td>
<td>$0.686 \pm 0.026$</td>
<td>10</td>
</tr>
<tr>
<td>honeycomb</td>
<td>$0.08722 \pm 0.00204$</td>
<td>$0.720 \pm 0.161$</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 6.1: Critical hopping $t_c$ and critical exponent $\nu$ of the $n = 1$ Mott state for different simple lattice geometries estimated from $N_{\text{Padés}}$ non-defective DLog-Padés of order $n \in \{9,10,11\}$.

The series of the gap $\Delta(t/U)$ for the individual lattices are found in eqs. (B.1)–(B.3) of the appendix. For the triangular and square lattice we verify the series previously obtained by Elstner and Monien [37, 114] employing the same method. From the series we estimate the critical coupling $t_c$ and critical exponent $\nu$ (table 6.1) using the $9^{th}$, $10^{th}$ and $11^{th}$ order DLog-Padé approximants of the series.
6.5 Results

Table 6.2: Critical hopping $t_c$ and critical exponent $\nu$ of the $n = 2$ Mott state for different simple lattice geometries estimated from $N_{\text{P}ad\acute{e}}$ non-defective Padés of order $n \in \{6, 7, 8\}$.

<table>
<thead>
<tr>
<th>Lattice</th>
<th>$t_c/U$</th>
<th>$\nu$</th>
<th>$N_{\text{P}ad\acute{e}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>triangular</td>
<td>$0.02229 \pm 0.00003$</td>
<td>$0.687 \pm 0.007$</td>
<td>17</td>
</tr>
<tr>
<td>square</td>
<td>$0.03522 \pm 0.00042$</td>
<td>$0.686 \pm 0.050$</td>
<td>14</td>
</tr>
<tr>
<td>honeycomb</td>
<td>$0.05120 \pm 0.00150$</td>
<td>$0.682 \pm 0.134$</td>
<td>17</td>
</tr>
</tbody>
</table>

Tables 6.1 and 6.2 list the critical values for the transition from the Mott $n = 1$ and Mott $n = 2$ state to the superfluid phase for the individual lattice geometries. The estimates are the mean value of the non-defective DLog-Padé approximants, the reported error estimates denote the maximum absolute deviation from the corresponding mean values. The last column of the tables indicate the number of DLog-Padé approximants considered to obtain the estimate. In some cases, more than half of the approximants were discarded as “defective” due to spurious root-pole pairs in the physical disc $|x| < t_c/U$ or confluent singularities close to the critical point itself.

As naïvely expected, the critical hopping $t_c$ is roughly inversely proportional to the coordination number of the lattice. Aside from the previous series expansion results, the obtained values are also in excellent agreement with studies using Quantum Monte Carlo reporting $t_c/U = 0.061 \pm 0.003$ [116] and $t_c/U = 0.05974(3)$ [117] for the square lattice, and the perturbative process-chain approach yielding $t_c/U = 0.03759$ for the triangular and $t_c/U = 0.08628$ for the honeycomb lattice [118]. The estimates also agree well with analytic findings from a cluster Gutzwiller approach, which predicts $t_c/U = 0.0596(4)$ for the square and $t_c/U = 0.0870(5)$ for the honeycomb lattice [103].

Regardless of the lattice geometry and the particular Mott state, the critical exponent estimate of the closing gap is $\nu \approx 0.685$. While the estimate for honeycomb lattice deviates from this value, one has to bear in mind that this particular estimate is subject to large errors, as it was obtained from only 7 approximants with a wide spread $\delta t_c = 0.00204$, $\delta \nu = 0.161$. Compared to the known critical exponent of the associated universality class [93] of the three-dimensional XY model $\nu = 0.67155(27)$ [119], the estimates slightly overestimate the expected value.

6.5.2 Stacked Lattices

For the stacked lattices we computed the $10^{th}$ order series for ground state and effective Hamiltonians of the $n = 1$ Mott state and the $8^{th}$ order series for the Mott $n = 2$ state. The series feature two independent perturbation parameters
for the intra-layer hopping \( t \) and the inter-layer hopping \( t' \). This allows to continuously tune the system from purely two-dimensional lattices \( t' = 0 \) to full three-dimensional lattices \( t' = t \). While the series are also valid in the one-dimensional limit of disconnected chains \( t = 0, t' \neq 0 \), and could be used to investigate the crossover from the BKT transition of the one dimensional system \([93, 100, 37]\) to the continuous phase transition of the three dimensional lattice, we focus on the two- to three-dimensional cases here.

Figure 6.4: Stacked-triangular lattice: Dispersion of particle (\( \omega^p(k) \), upper band) and hole (\( -\omega^h(k) \), lower band) quasi-particle on the \( n = 1 \) Mott state close to the critical point \( t_c \) of the isotropic case \( t'/t = 1 \). The path through the Brillouin zone is shown on the left. Light shaded regions are qualitative error estimates.

Figures 6.4–6.6 show the dispersion along the depicted path in the first Brillouin zone for different inter-layer hoppings \( t' \in [0, t] \). All dispersions are calculated for \( t \) slightly below the critical hopping \( t_c \) of the respective isotropic case \( t = t' \). The error-regions represent the spread of the Padé approximants leading to the dispersion line and are qualitative estimates, rather than quantitative errors. Without any inter-layer coupling \( t' = 0 \) the dispersions orthogonal to the layers are flat as expected. The minima forming the energy gap are straight lines through the \( \Gamma \)-point along the same direction.

Ramping up the inter-layer coupling the dispersions of all lattice geometries reduce the energy close to the lattice plane \( k_z = 0 \) and increase the energy towards \( k_z = \pi \). The effect sets in very slowly with \( t' \) and is barely visible for \( t'/t \leq 0.1 \). Due to the shift of energies also the degeneracy of the energy gap is lifted and
unique minima form at the Γ-point. For \( t = t' \) the gap \( \Delta = E^p - E^h \) almost closes, since the \( t \) was chosen slightly below its critical value for this case.

While those observations apply to all stacked-lattices, the dispersion of the (stacked-)honeycomb lattice displays some additional notable features. Due to the two sites per unit cell in the hexagonal lattice the particle and hole dispersion splits into two modes touching at the \( K \) points and the lines along \( k_z \) intersecting these points and bear a striking resemblance to the band structure of graphene, where the linear dispersion at the \( K \) points leads to massless Dirac fermions [111].

The phase diagrams are computed from the series of the particle and hole energy minima through plain Padé approximation. Each non-defective Padé is shown with a semi-transparent line, such that distribution of the individual approximants is visible. Regions where many approximants agree appear darker in the plots. For the stacked-square lattice we have compared two limits, namely the 2d square lattice \( (t' = 0) \) and the isotropic cubic lattice \( (t' = t) \), with previous Quantum Monte Carlo (QMC) [117, 120] and bosonic dynamic mean-field theory (B-DMFT) [101] calculations (fig. 6.7). Overall, we find good agreement of the phase diagram predictions. While the Padé approximants start to spread close to the tip of the Mott lobes they remain between the B-DMFT and QMC results up to the direct vicinity of the tip. At the tip itself the plain Padé approx-
imants overestimate $t_c$, in contrast to the other methods. The reason for this is two-fold: first, the results of the perturbation become less reliable with increasing $t/U$ which is reflected qualitatively by the spread of the approximants. Second and more importantly, it is the effect of a deficiency of plain Padé approximants: as rational functions, their functional form does not allow proper representation of the power-law behavior expected in proximity of the critical point. DLog-Padé approximants being tailored for this functional behavior, on the other hand, will exhibit higher curvature close to $t_c$ and capture the critical behavior more accurately. Therefore, we employ plain Padés only for the phase boundary diagram and compute the estimates of the critical hoppings $t_c$ from DLog-Padé approximants. While approximating the phase boundary from DLog-Padés by numerical integration for each point is also feasible, the employed plain Padés already yield accurate results for the majority of the diagram and their application serves as an illustrative example of their shortcomings.

The critical values (table 6.4) for the $t' = 0$ case are consistent with our findings from higher-order series for the square lattice in the previous section, but are subject to larger errors. For the isotropic case $t'/t = 1$ the critical coupling $t_c/U = 0.03418 \pm 0.00023$ is in remarkable agreement with known results from QMC simulations, $t_c/U = 0.03408(2)$ [120], and the perturbative approach by Teichmann.
Figure 6.7: Phase diagram comparison of the square lattice $t'/t = 0$ limit and the cubic lattice $t'/t = 1$ limit obtained from Padé approximation with results from QMC [117, 120] and B-DMFT [101] simulations.

et al. $t_c/U = 0.03407 \pm 0.0003$ [121]. The value is also consistent with an analytic approach using a cluster Gutzwiller method predicting $t_c/U = 0.0350(2)$ [103].

The critical exponent $\nu = 0.567$ for the isotropic limit of the stacked-square lattice overestimates the expected mean-field value $\nu = 1/2$ from the (d+1)-dimensional XY model. The mean-field theory result is exact in this case, as four is the upper critical dimension for the model [122]. Also for the square lattice limit, the $10^{th}$ order series lead to a substantially higher estimate of the critical exponent $\nu = 0.698$ compared to the $12^{th}$ order series $\nu = 0.686$. Therefore the large deviation of the expected value of the cubic limit seems to be an artifact of the reduced series order rather than the additional dimension.

Comparing the critical behavior (tables 6.3–6.5) and phase diagrams (figs. 6.8–6.10) of the different stacked lattices, the qualitative behavior may be understood by looking at the relative increase of connectivity of a single site when switching on the inter-layer coupling. While the critical hopping $t$ of the stacked-honeycomb lattice shrinks to half $t_c/U = 0.043$ of the two-dimensional value $t_c/U = 0.087$, the critical hopping $t_c/U = 0.025$ of the stacked-triangular is only reduced to 65% of the original value for the triangular lattice $t_c/U = 0.038$. For all lattices the critical hopping $t_c$ and critical exponent $\nu$ change smoothly when turning on the
Table 6.3: Stacked Triangular lattice: critical hopping $t_c$ and critical exponent $\nu$ of the $n = 1$ Mott state for different inter-plane hoppings $t'_c$ from Padé approximants of order $n \in \{7, 8, 9\}$. For $t'/t = 0.1$ two estimates were obtained, where the estimate marked † excludes nine border case Padé approximants, which feature a “spurious” singularity in the vicinity of $t_c$ ($t \approx 0.0446$) and could be considered defective.

<table>
<thead>
<tr>
<th>$t'/t$</th>
<th>$t_c/U$</th>
<th>$\nu$</th>
<th>$N_{\text{Padé}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.03780 ± 0.00005</td>
<td>0.686 ± 0.006</td>
<td>13</td>
</tr>
<tr>
<td>0.01</td>
<td>0.03744 ± 0.00004</td>
<td>0.671 ± 0.006</td>
<td>8</td>
</tr>
<tr>
<td>0.1†</td>
<td>0.03496 ± 0.00006</td>
<td>0.606 ± 0.010</td>
<td>8</td>
</tr>
<tr>
<td>0.1</td>
<td>0.03485 ± 0.00014</td>
<td>0.583 ± 0.028</td>
<td>17</td>
</tr>
<tr>
<td>0.5</td>
<td>0.02908 ± 0.00002</td>
<td>0.562 ± 0.004</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>0.02497 ± 0.00003</td>
<td>0.559 ± 0.009</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 6.4: Stacked Square lattice: critical hopping $t_c$ and critical exponent $\nu$ of the $n = 1$ Mott state for different inter-plane hoppings $t'_c$ from Padé approximants of order $n \in \{7, 8, 9\}$.

<table>
<thead>
<tr>
<th>$t'/t$</th>
<th>$t_c/U$</th>
<th>$\nu$</th>
<th>$N_{\text{Padé}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.05987 ± 0.00080</td>
<td>0.698 ± 0.076</td>
<td>19</td>
</tr>
<tr>
<td>0.01</td>
<td>0.05896 ± 0.00073</td>
<td>0.676 ± 0.068</td>
<td>17</td>
</tr>
<tr>
<td>0.1</td>
<td>0.05311 ± 0.00052</td>
<td>0.580 ± 0.054</td>
<td>21</td>
</tr>
<tr>
<td>0.5</td>
<td>0.04156 ± 0.00026</td>
<td>0.565 ± 0.028</td>
<td>19</td>
</tr>
<tr>
<td>1</td>
<td>0.03418 ± 0.00023</td>
<td>0.567 ± 0.031</td>
<td>12</td>
</tr>
</tbody>
</table>

inter-layer coupling. For $t' = 0.01t$ we observe only a marginal decrease of the values and the phase diagram remains indistinguishable from the two-dimensional case (not shown). Within error estimates the critical exponent $\nu$ seems to change in a similar fashion when ramping up the third dimension, regardless of the lattice geometry of the layers. However, for the $10^{th}$ order series the qualitative error estimates on the critical exponent are rather large, thus higher accuracy results are required to substantiate this claim.

### 6.6 Extensibility

The application for the perturbative $T = 0$ series expansion of the Bose-Hubbard used to derive the presented series can easily be extended to include nearest-neighbor repulsion $V \sum_{\langle i,j \rangle} n_i n_j$, where numerical evidence for additional solid and
6.6 Extensibility

<table>
<thead>
<tr>
<th>$t'/t$</th>
<th>$t_c/U$</th>
<th>$\nu$</th>
<th>$N_{\text{Padés}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$0.08676 \pm 0.00307$</td>
<td>$0.685 \pm 0.168$</td>
<td>18</td>
</tr>
<tr>
<td>0.01</td>
<td>$0.08482 \pm 0.00268$</td>
<td>$0.660 \pm 0.147$</td>
<td>19</td>
</tr>
<tr>
<td>0.1</td>
<td>$0.07370 \pm 0.00074$</td>
<td>$0.580 \pm 0.044$</td>
<td>18</td>
</tr>
<tr>
<td>0.5</td>
<td>$0.05393 \pm 0.00029$</td>
<td>$0.574 \pm 0.026$</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>$0.04258 \pm 0.00023$</td>
<td>$0.571 \pm 0.027$</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 6.5: Stacked Honeycomb lattice: critical hopping $t_c$ and critical exponent $\nu$ of the $n = 1$ Mott state for different inter-plane hoppings $t'_c$ from Padé's of order $n \in \{7, 8, 9\}$.

Figure 6.8: Phase diagram of the stacked triangular lattice. Hopping $t$ in units of $t_c = 0.02497U$.

...supersolid phases exists [96, 97, 123], or different sublattice potentials as suggested for the honeycomb lattice in [99]. As the application is based on the same framework of generic algorithms as the high-temperature series expansion application for Heisenberg models of the previous chapter, applications for high-order perturbation theory on entirely different models, such as dimerized spin models, $t$-$J$ models or Fermi-Hubbard models on arbitrary lattices can be realized by implementing the respective operators $H_0$ and $V$ with only little additional effort.

88
Figure 6.9: Phase diagram of the stacked square lattice. Hopping $t$ in units of $t_c = 0.03418U$.

6.7 Conclusion and Outlook

We presented a set of computer programs to compute the ground-state phase diagram and critical properties of the Mott states of the Bose-Hubbard model on arbitrary lattices. To demonstrate the possibilities of the programs, we computed the phase diagrams, showing the boundaries of the commensurate $n = 1$ and $n = 2$ Mott states for the two-dimensional triangular, square, and honeycomb lattices, as well as the three-dimensional weakly coupled stacked lattices thereof. Additionally, we extracted estimates for the critical hopping $t_c/U$ and the critical exponent $\nu$ of the phase transition to the superfluid phase of at the tip of the $n = 1$ Mott lobe from the series of the particle-hole energy gap. Comparison of limiting cases with previous results obtained with other methods reveal excellent agreement with our findings.

Since all of the mentioned lattices can nowadays be experimentally realized with optical lattices [109, 124], and the Mott-insulator-superfluid transition has been observed in some of those lattices, our results may serve as a map for experimentalists when tuning their lattices or studying the phase transition. Furthermore, as the presented programs allow to compute the phase diagram for arbitrary regular lattices, they may provide easily accessible benchmarks for future optical lattice setups, where increasingly complex two- and three-dimensional lattice geometries
Figure 6.10: Phase diagram of the stacked honeycomb lattice. Hopping $t$ in units of $t_c = 0.04258 U$

become feasible.
Chapter 7

Conclusion

In this thesis we presented two series expansion methods for quantum lattice models: high-temperature series expansions and perturbative series expansions at $T = 0$.

We developed a flexible C++ framework providing generic building-blocks to easily implement such expansions for arbitrary lattice models. Based on the framework we present computer programs to calculate high-temperature series for spin-$1/2$ Heisenberg models and perturbative $T = 0$ expansions for Bose-Hubbard models.

An essential part of the series expansions is the linked-cluster method, which relies heavily on graph theoretical algorithms to determine if graphs are isomorphic and to generate graphs which are embeddable into a specific lattice. The required algorithms were implemented in the ALPS project [1]. Here we explained the underlying McKay algorithm and extended it to graphs with colored edges.

For efficient series calculations on today’s supercomputers, we presented a high-performance computing (HPC) C++ library for fast fixed high-precision integers with up to 512 bit and statically sized polynomials in up to four symbolic variables. The computational hot spot of high-temperature series expansions are inner products of vectors of polynomials with high-precision integer coefficients. For this special operation the library provides a hybrid CPU-GPU function, which takes advantage of the graphical processing units (GPUs) built into today’s supercomputers. Benchmarks against the standard GNU Multiple Precision (GMP) library revealed a speedup of 4.2x for plain integer addition and 2.5x for plain integer multiplication. For the inner product we achieved a speedup of up to 13x compared to a naïve solution based on GMP and OpenMP.

The high-temperature series expansion programs for the spin-$1/2$ Heisenberg model allow to compute series for the free energy, the uniform magnetic susceptibility at zero field, and the static magnetic structure factor on arbitrary lattices with up to four independent coupling constants $J_1, \ldots, J_4$. We studied the aniso-
tropic triangular lattice and provided a step-by-step guide how to extract estimates for the coupling constants by fitting Padé approximants to experimental susceptibility data. For the non-frustrated case with antiferromagnetic couplings on the distorted square sublattice and ferromagnetic coupling on the diagonal chain sublattice we found excellent agreement with Quantum Monte Carlo simulations. From magnetic susceptibility measurements on Cs$_2$CuBr$_4$ and the ordering wave vector obtained from neutron-scattering experiments, we could estimate the antiferromagnetic coupling constants to be $J_1 = 4.9$K and $J_2 = 15.2$K ($J_1/J_2 = 0.3(2)$) for the fully frustrated system, where $J_1$ is the coupling of the distorted square sublattice and $J_2$ the coupling along the diagonal chains.

The perturbative $T = 0$ series expansion programs for the Bose-Hubbard model enables high-order Rayleigh-Schrödinger perturbation series of the ground state energy of a Mott state about the atomic limit, as well as deriving perturbative effective Hamiltonians for particle and hole excitations of the Mott state. Also these computer programs are able to derive the series for arbitrary lattices with up to four independent hopping parameters $t_1, \ldots, t_4$.

We computed the perturbative series for the triangular lattice, the square lattice, the honeycomb lattice, and stacked layers of each lattice geometry, where we considered independent inter-layer and intra-layer couplings. For each lattice we computed the effective Hamiltonians for the particle and hole excitations on the $n = 1$ and $n = 2$ Mott states and obtained the dispersion of the dressed particle and hole. From the minimal energy of the particle and hole dispersions we could map out the phase boundary of the Mott phases. For the square lattice we found remarkable agreement between the phase boundary predicted by the perturbation series, Quantum Monte Carlo and B-DMFT. We also extracted the critical hopping $t_c$ at the tip of the Mott lobes and determined the associated critical exponent $\nu$ of the closing energy gap. The critical exponent obtained for the two-dimensional lattices $\nu \approx 0.685$ slightly overestimated the expected three-dimensional XY model universality class value $\nu = 0.67155(27)$. For the three-dimensional stacked lattices the difference of our estimate $\nu \approx 0.571$ to the expected universality class value $\nu = 1/2$ is worse. However, we attribute this mainly to the lower expansion orders for the stacked lattices.

The high-temperature series expansion programs for the Heisenberg model, as well as the perturbative series expansion programs for the Bose-Hubbard system, may serve as powerful tools in the lab. The high-temperature series for Heisenberg models offer a valuable link between the microscopic model parameters and the macroscopic measurements on real materials. The laid-out method to extract the coupling constants may thereby help to gain a better understanding of experimental results directly in the research labs. For Bose-Hubbard models, the effective Hamiltonians and phase boundaries accessible through the program can be used as benchmarks for future optical lattice setups, where increasingly complex lattices
are realized.

Given the flexibility of the developed generic C++ framework for series expansions many other lattice models are within reach and require only the definition of new operators. Of particular interest may be fermionic models such as the $t$-$J$ model or the more general Hubbard model, where series expansions may obtain competitive accuracy with current methods [125]. Another interesting candidate may be the Heisenberg-Kitaev model on a honeycomb lattice, which describes iridates where strong spin-orbit coupling leads to novel exotic phases [126, 127].
Appendix A

Computer Programs

The source code of the computer programs for

- high-temperature series expansions for the Heisenberg model
- perturbative series expansions at $T = 0$ for the Bose-Hubbard model

including a generic C++ framework for series expansions, which allows for easy extension to other models, and all Mathematica notebooks used for the analysis in chapters 5 and 6 are available for download at http://www.comp-phys.org/lcse.

The graph library, for detecting isomorphisms, embedding and graph generation for lattices is part of the ALPS libraries [1]. Within the source code of the ALPS, the graph library is found in src/alps/graph. ALPS is available for download at http://alps.comp-phys.org.

The source code of the VLI library is available at http://www.comp-phys.org/vli.
Appendix B

Series data

For further reference we list the most important series derived in this thesis. Unfortunately we cannot present the $k$-resolved quantities here, due the sheer number of terms. All series can be found in the Mathematica notebooks (see appendix A) or the files referenced therein.

B.1 High-Temperature Series of the Heisenberg Model

The series coefficients for the magnetic zero-field susceptibility $\chi$ of the spin-1/2 Heisenberg model is shown in table B.1.
## B.1 High-Temperature Series of the Heisenberg Model

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>$a_{n,m}$</th>
<th>$n$</th>
<th>$m$</th>
<th>$a_{n,m}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>4</td>
<td>0</td>
<td>+13/192</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1/2</td>
<td>4</td>
<td>1</td>
<td>-17/192</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>+1/24</td>
<td>4</td>
<td>2</td>
<td>+581/15360</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>+5/384</td>
<td>4</td>
<td>3</td>
<td>-1343/61440</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>-7/1280</td>
<td>4</td>
<td>4</td>
<td>+39833/1032192</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>-133/30720</td>
<td>4</td>
<td>5</td>
<td>-34151/1720320</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>+1/4032</td>
<td>4</td>
<td>6</td>
<td>-654169/61931520</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>+1269/146880</td>
<td>4</td>
<td>7</td>
<td>+123366899/14863564800</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>+3737/18579456</td>
<td>4</td>
<td>8</td>
<td>+3060454129/653996851200</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>-339691/1486356480</td>
<td>5</td>
<td>0</td>
<td>-71/1920</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>-1428209/13624934400</td>
<td>5</td>
<td>1</td>
<td>+191/1920</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>+18710029/560568729600</td>
<td>5</td>
<td>2</td>
<td>-3419/46080</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>5</td>
<td>3</td>
<td>-37/6864</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>+1</td>
<td>5</td>
<td>4</td>
<td>+204653/5160960</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-1/4</td>
<td>5</td>
<td>5</td>
<td>+3095149/309657600</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>-1/12</td>
<td>5</td>
<td>6</td>
<td>-33720647/1238630400</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>+1/64</td>
<td>5</td>
<td>7</td>
<td>-95689567/18166579200</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>+23/960</td>
<td>6</td>
<td>0</td>
<td>-7/36864</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>+67/46080</td>
<td>6</td>
<td>1</td>
<td>-219/10240</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>-1271/215040</td>
<td>6</td>
<td>2</td>
<td>+1129/26880</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>-361/215040</td>
<td>6</td>
<td>3</td>
<td>-69791/15482880</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>+22433/18579456</td>
<td>6</td>
<td>4</td>
<td>+11687/2903040</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>+892291/1238630400</td>
<td>6</td>
<td>5</td>
<td>+4785047/206438400</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>-2696291/16349921280</td>
<td>6</td>
<td>6</td>
<td>-1201172057/326998425600</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>+1/2</td>
<td>7</td>
<td>0</td>
<td>+811/322560</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-1/16</td>
<td>7</td>
<td>1</td>
<td>-1369/61440</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>+25/64</td>
<td>7</td>
<td>2</td>
<td>+30497/737280</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-1/16</td>
<td>7</td>
<td>3</td>
<td>-292231/37158912</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>+151/3840</td>
<td>7</td>
<td>4</td>
<td>-60070847/1486356480</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>+77/9216</td>
<td>7</td>
<td>5</td>
<td>+3295644479/163499212800</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>+29119/2580480</td>
<td>8</td>
<td>0</td>
<td>+8213/5160960</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>-14033/10321920</td>
<td>8</td>
<td>1</td>
<td>-61993/10321920</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>-165209/46448640</td>
<td>8</td>
<td>2</td>
<td>-23951/123863040</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>-30397/530841600</td>
<td>8</td>
<td>3</td>
<td>+19321523/928972800</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>+8128031/7431782400</td>
<td>8</td>
<td>4</td>
<td>-3477829/185794560</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-1/6</td>
<td>9</td>
<td>0</td>
<td>-1729/829440</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>+19/96</td>
<td>9</td>
<td>1</td>
<td>+3175/23224320</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>-1/6</td>
<td>9</td>
<td>2</td>
<td>-249945907/7431782400</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>+1607/11520</td>
<td>9</td>
<td>3</td>
<td>+863358541/30656102400</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>-251/6144</td>
<td>10</td>
<td>0</td>
<td>-2514101/3715891200</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>-2279/71680</td>
<td>10</td>
<td>1</td>
<td>+38886349/7431782400</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>+92311/7741440</td>
<td>10</td>
<td>2</td>
<td>-50084021/5945425920</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>+2375329/185794560</td>
<td>11</td>
<td>0</td>
<td>+27560147/40874803200</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>-1856821/7431782400</td>
<td>11</td>
<td>1</td>
<td>-348043/6386680</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>-483818413/98099527680</td>
<td>12</td>
<td>0</td>
<td>+870952109/1961990553600</td>
</tr>
</tbody>
</table>

Table B.1: Series coefficients of \( \chi/\beta = \frac{1}{4} \sum_{n,m} a_{n,m}(\beta J_1)^n(\beta J_2)^m \) for the Heisenberg model on the anisotropic triangular lattice.
B.2 Perturbation Series at $T = 0$ of the Bose-Hubbard Model

The following list shows the series of the energy gap $\Delta_1$ separating the particle and hole bands at $k = 0$ on a $n = 1$ Mott state for the various lattices. The gap series for the stacked lattices are shown in tables B.2–B.4.

- Triangular lattice

\[
\Delta^{\text{tri}}_1(X) = 1 - 18X - 81X^2 - 819X^3 - \frac{54891}{4}X^4 - \frac{11459377}{50}X^5
- \frac{6764830501}{1500}X^6 - \frac{3957593443549}{45000}X^7
- \frac{263600338160273}{140000}X^8 - \frac{79054659543137812691}{198450000}X^9
- \frac{49507676563116513700717}{248109135750000000}X^{10}
- \frac{388956134470706053612905963}{55566000000}X^{11}
- \frac{112803369695417592352577307407077}{2481091357500000000}X^{12} \quad \text{(B.1)}
\]

- Square lattice

\[
\Delta^{\text{sq}}_1(X) = 1 - 12X - 22X^2 - 264X^3 - \frac{15659}{10}X^4 - \frac{656984}{25}X^5
- \frac{513002341}{2250}X^6 - \frac{4465455218}{1125}X^7 - \frac{2194497431888101}{56700000}X^8
- \frac{523244582353596437}{744187500}X^9 - \frac{4749112579154967367231}{625117500000}X^{10}
- \frac{6676218845916748474723399}{49228003125000}X^{11}
- \frac{13227933428099377964624765709673}{8196014797500000000}X^{12} \quad \text{(B.2)}
\]
B.2 Perturbation Series at $T = 0$ of the Bose-Hubbard Model

- Honeycomb lattice

\[
\Delta_{hc}^2(X) = 1 - 9X - \frac{9}{2}X^2 - \frac{117}{2}X^3 - \frac{21513}{40}X^4 - \frac{793067}{200}X^5 - \frac{6829763}{600}X^6
- \frac{2976150863}{15000}X^7 - \frac{32189735960147}{10800000}X^8
- \frac{3038973213234487}{16200000}X^9 - \frac{1850978523968117671}{11340000000}X^{10}
- \frac{37317540258992730821489}{3214890000000}X^{11}
- \frac{845301414595631798429657071}{27848984625000000}X^{12}
\]  
\]  
\]  
\] (B.3)
Table B.2: Series coefficients of single particle gap $\Delta^{s\text{-tri}}(X, K) = \sum_{n,m} a_{n,m} X^n K^m$ of the Bose-Hubbard model on a stacked triangular lattice, where $X = t/U$ and $K = t'/U$.  

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>$a_{n,m}$</th>
<th>$n$</th>
<th>$m$</th>
<th>$a_{n,m}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>-34114863119/30000</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-6</td>
<td>2</td>
<td>5</td>
<td>-7495269397/900</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>-1729907746909/37500</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>7</td>
<td>-39283590776741/14625</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>287/20</td>
<td>2</td>
<td>8</td>
<td>-78397584192379630313/51030000000</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5821/50</td>
<td>3</td>
<td>0</td>
<td>-819</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>-602243/1000</td>
<td>3</td>
<td>1</td>
<td>-50556</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>8186059/2500</td>
<td>3</td>
<td>2</td>
<td>-6848871/10</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>-99230204543/5400000</td>
<td>3</td>
<td>3</td>
<td>-751997054/75</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>1466604668323/1215000000</td>
<td>3</td>
<td>4</td>
<td>-994066863601/11250</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>-6295148943458549/7290000000</td>
<td>3</td>
<td>5</td>
<td>-24855001065762/28125</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-18</td>
<td>3</td>
<td>6</td>
<td>-1791571329855573/28125</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-96</td>
<td>3</td>
<td>7</td>
<td>-207705019706685308/308671875</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-414</td>
<td>4</td>
<td>0</td>
<td>-54891/4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>-1944</td>
<td>4</td>
<td>1</td>
<td>-12171707/10</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>-384947/50</td>
<td>4</td>
<td>2</td>
<td>-72730643671/3000</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>-12464004/375</td>
<td>4</td>
<td>3</td>
<td>-115922453313/2500</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>-30165111/2500</td>
<td>4</td>
<td>4</td>
<td>-51325897333817417/9450000</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>-50088176761/84375</td>
<td>4</td>
<td>5</td>
<td>-6711885630139055263/9922500</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>-1088685334539/8100000</td>
<td>4</td>
<td>6</td>
<td>-24156695728537004253677/39609843750</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>-2065345014833759/151875000</td>
<td>5</td>
<td>0</td>
<td>-11459377/50</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-81</td>
<td>5</td>
<td>1</td>
<td>-3762417962/125</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-2142</td>
<td>5</td>
<td>2</td>
<td>-5817709084321/7500</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-94959/5</td>
<td>5</td>
<td>3</td>
<td>-210306172972463/84375</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-4520134/25</td>
<td>5</td>
<td>4</td>
<td>-91045226579725453613/330750000</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>-4520134/25</td>
<td>5</td>
<td>5</td>
<td>-2749154929060810363763/6511646025</td>
</tr>
</tbody>
</table>

Series data
### B.2 Perturbation Series at $T = 0$ of the Bose-Hubbard Model

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>$a_{n,m}$</th>
<th>$n$</th>
<th>$m$</th>
<th>$a_{n,m}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>-22</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-6</td>
<td>2</td>
<td>1</td>
<td>-852</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>-33146/5</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>-1773956/25</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>287/20</td>
<td>2</td>
<td>4</td>
<td>-361170107/900</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5821/50</td>
<td>2</td>
<td>5</td>
<td>-36434740879/11250</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>-602243/1000</td>
<td>2</td>
<td>6</td>
<td>-2783122262039/16875</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>8186059/2500</td>
<td>2</td>
<td>7</td>
<td>-4024916029468358/3796875</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>-992394254453/5400000</td>
<td>2</td>
<td>8</td>
<td>-4420179922437605233/76545000000</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>14666046488323/1215000000</td>
<td>3</td>
<td>0</td>
<td>-264</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>-6295148943458549/7290000000</td>
<td>3</td>
<td>1</td>
<td>-13360</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-12</td>
<td>3</td>
<td>2</td>
<td>-154532456/75</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-64</td>
<td>3</td>
<td>3</td>
<td>-3074127856/1125</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-276</td>
<td>3</td>
<td>4</td>
<td>-44519405706/16875</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>-1296</td>
<td>3</td>
<td>5</td>
<td>-2046409523762/84375</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>-384947/75</td>
<td>3</td>
<td>6</td>
<td>-143423077065103/759375</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>-24928012/1125</td>
<td>3</td>
<td>7</td>
<td>-82053311570679038/56953125</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>-301261511/3750</td>
<td>4</td>
<td>0</td>
<td>-15659/10</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>-15017635522/253125</td>
<td>4</td>
<td>1</td>
<td>-4915347/25</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>-1086865345439/121500000</td>
<td>4</td>
<td>2</td>
<td>-16031887439/4500</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>-208534501483375/9227812500</td>
<td>4</td>
<td>3</td>
<td>-8680929806539/11250</td>
</tr>
</tbody>
</table>

**Table B.3:** Series coefficients of single particle gap $\Delta_{s-sq}(X, K) = \sum_{n,m} a_{n,m} X^n K^m$ of the Bose-Hubbard model on a stacked square lattice, where $X = t/U$ and $K = t'/U$. 

102
Table B.4: Series coefficients of single particle gap $\Delta_{s-hc}^n(X,K) = \sum_{n,m} a_{n,m} X^n K^m$ of the Bose-Hubbard model on a stacked honeycomb lattice, where $X = t/U$ and $K = t'/U$. 

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>$a_{n,m}$</th>
<th>$n$</th>
<th>$m$</th>
<th>$a_{n,m}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>-35130788/375</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-6</td>
<td>3</td>
<td>4</td>
<td>-800414557489/90000</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>-56262565813/6750</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>-4334829932249/67500</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>287/20</td>
<td>3</td>
<td>7</td>
<td>-789539925630791641/1594687500</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5821/50</td>
<td>4</td>
<td>0</td>
<td>-21513/30</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>-602243/1000</td>
<td>4</td>
<td>1</td>
<td>-4599769/100</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>8186059/2500</td>
<td>4</td>
<td>2</td>
<td>-93628401/100</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>-9923020453/54000000</td>
<td>4</td>
<td>3</td>
<td>-108585427942/5625</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>14660046468323/12150000</td>
<td>4</td>
<td>4</td>
<td>-1340653858142201/6300000</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>-6295148914456849/72900000000</td>
<td>4</td>
<td>5</td>
<td>-95439068410720222/33075000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-9</td>
<td>4</td>
<td>6</td>
<td>-61374103553491221616539/2500470000000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-48</td>
<td></td>
<td></td>
<td>-793807/200</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-207</td>
<td></td>
<td></td>
<td>-129249261/250</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>-972</td>
<td></td>
<td></td>
<td>-11650950113/7500</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>-384947/100</td>
<td></td>
<td></td>
<td>-31242342083158/84375</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>-6252003/375</td>
<td></td>
<td></td>
<td>-11530477292746628809/1984500000</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>-3012611/5000</td>
<td></td>
<td></td>
<td>-4434108835092990094320/5209312500</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>-50088176761/1687500</td>
<td></td>
<td></td>
<td>-6829763/600</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>-10868533345439/16200000</td>
<td></td>
<td></td>
<td>-962525459/180</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>-2065345014833759/3037500000</td>
<td></td>
<td></td>
<td>-176212651810493/1050000</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-9/2</td>
<td></td>
<td></td>
<td>-1803925784999251237/29707500</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>423</td>
<td></td>
<td></td>
<td>-2100096140286434023998721432600000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-27099/10</td>
<td></td>
<td></td>
<td>-2976150663/150000</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-865667/25</td>
<td></td>
<td></td>
<td>-1676650707892/28125</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>-1001344243/6000</td>
<td></td>
<td></td>
<td>-270471087113702587/992250000</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>-70396006993/450000</td>
<td></td>
<td></td>
<td>-3887131715874776034087/39069843750</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>-106221451513/150000</td>
<td></td>
<td></td>
<td>-321897359369147/10800000</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>-2084875728598/421875</td>
<td></td>
<td></td>
<td>-3470396502554107/54000000</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>-271039778476692699/102060000000</td>
<td></td>
<td></td>
<td>-26993477598737018566667/833490000000</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-117/2</td>
<td></td>
<td></td>
<td>-3038973213234487/16200000</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4440</td>
<td></td>
<td></td>
<td>-94341987426751607189/12757500000</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>-3403581/50</td>
<td></td>
<td></td>
<td>-185097852968117671/113400000000</td>
</tr>
</tbody>
</table>

Series data
B.2 Perturbation Series at $T = 0$ of the Bose-Hubbard Model


List of publications


Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


[87] Magnetic susceptibility of Cs$_2$CuBr$_4$ measured by Natalija van Well at Laboratory for Neutron Scattering and Imaging, Paul Scherrer Institute, Switzerland


Acknowledgments

First I would like to thank my supervisor Matthias Troyer for giving me the unique opportunity to pursue PhD in his group, for introducing me to advanced C++ programming techniques and high-performance computing, and for his advice on how to write good papers and proposals. I would also like to thank Simon Trebst for inviting me to Station Q in the beginning of my studies, for being my co-examiner, and the many comments and corrections to my thesis.

I am grateful for the collaborations with Lukas Gamper and Timothée Ewart on the graph library and the very large integer library. Those libraries became large parts of my PhD project and are the result of many hours of discussing, coding, benchmarking, and hunting tricky bugs together.

What I enjoyed the most during my studies are the people I met in the international environment at ETH Zürich and the conferences. I would like to especially thank Vera, Tama, Jan, Alexey, Ethan, Damian, Troels, Lukas, Michele and Juan who significantly shaped the past few years and were a great support to me. Furthermore, I would like to thank Philippe, Brigitte, Bela, Lode, Peter, Jonathan, Matthias, Ruben, Juan-Carlos, Martin, Sebastiano, Lei, Itztok, Ilia, Schura, Kyril, Mauro, Hiroshi, the Ghostbusters, Jana, Medha, Thomas, Giuseppe, Vojtech, Guglielmo, Georg, QuanSheng, Jakub, Sergei, Zhenya, Mario, Slava, Donjan, Dominic, Bettina, Tao, Zhe, Sultan, Christian, and Wes. I am grateful for the countless discussions we had about physics, computers, and life during coffee breaks and Whisky Fridays.

Last and most importantly, I thank my parents and family for their unwavering support. I am deeply grateful that I could rely on them whenever I needed.
Als in der Frühe Eos erschien mit rosigen Fingern...

— Homer, *Odyssey*
(trasl.: K. Steinmann)