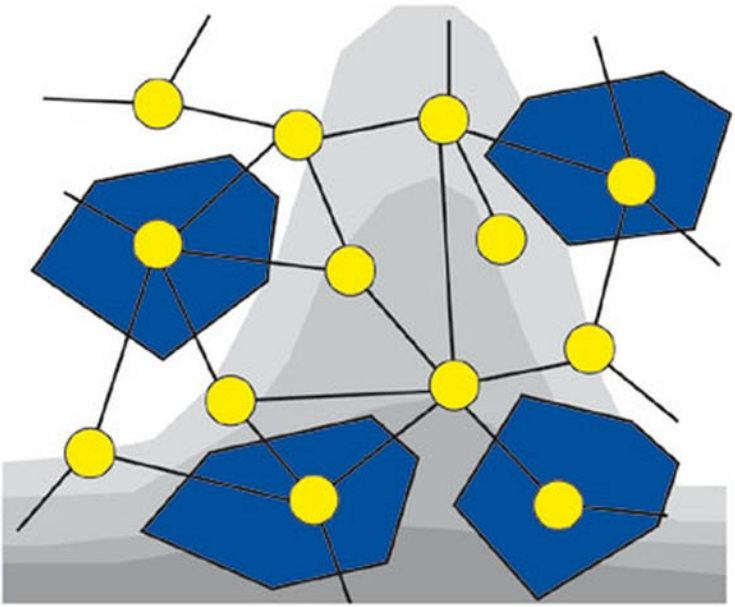
Alexander K. Hartmann
Martin Weigt

WILEY-VCH

Phase Transitions in Combinatorial Optimization Problems

Basics, Algorithms and Statistical Mechanics



Alexander K. Hartmann and Martin Weigt

Phase Transitions in Combinatorial Optimization Problems

Basics, Algorithms and Statistical Mechanics

WILEY-VCH Verlag GmbH & Co. KGaA

Alexander K. Hartmann
Martin Weigt

Phase Transitions in Combinatorial Optimization Problems

Basics, Algorithms and Statistical Mechanics

Related Titles

Hartmann, A. K. / Rieger, H. (eds.)
New Optimization Algorithms in Physics
2004. XII, 300 pages with 113 figures. Hardcover.
ISBN 3-527-40406-6

Gen, M. / Cheng, R.
Genetic Algorithms and Engineering Optimization
Series: Wiley Series in Engineering Design and Automation
2000. XVI, 496 pages. Hardcover.
ISBN 0-471-31531-1

Wolsey, L. A. / Nemhauser, G. L.
Integer and Combinatorial Optimization
Series: Wiley-Interscience Series in Discrete Mathematics and Optimization
1999. XVI, 764 pages. Softcover.
ISBN 0-471-35943-2

Chong, Edwin K. P. / Zak, Stanislaw H.
An Introduction to Optimization
Juli 2001, 496 pages. Hardcover.
ISBN 0-471-39126-3

Deb, Kalyanmoy
Multi-Objective Optimization Using Evolutionary Algorithms
Juni 2001, 518 pages. Hardcover.
ISBN 0-471-87339-X

Fletcher, Roger
Practical Methods of Optimization
Juni 2000, 450 pages. Softcover.
ISBN 0-471-49463-1

Alexander K. Hartmann and Martin Weigt

Phase Transitions in Combinatorial Optimization Problems

Basics, Algorithms and Statistical Mechanics

WILEY-VCH Verlag GmbH & Co. KGaA

The Authors of this Book

Dr. Alexander K. Hartmann
University of Goettingen,
Institute for Theoretical Physics
hartmann@theorie.physik.uni-goettingen.de

Dr. Martin Weigt
I.S.I. Institute for Scientific
Interchange Foundation, Turin
weigt@isiosf.isi.it

Library of Congress Card No.: applied for
British Library Cataloging-in-Publication Data:
A catalogue record for this book is available from the British Library.
Bibliographic information published by Die Deutsche Bibliothek.
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at http://dnb.ddb.de.

Printing Strauss GmbH, Mörlenbach

Binding Litges & Dopf Buchbinderei GmbH, Heppenheim

Printed in the Federal Republic of Germany

Printed on acid-free paper

ISBN-13: 978-3-527-40473-5
ISBN-10: 3-527-40473-2

Contents

Preface

This text book provides an introduction to the young and growing research area of statistical mechanics of combinatorial optimization problems. This cross-disciplinary field is positioned at the borderline between physics, computer science, and mathematics. Whereas most optimization problems and the majority of available computer algorithms used for their solution depend on computer science and mathematics, the main methods presented in this book are related to statistical physics.

Combinatorial optimization problems originate in real-world applications. The head of the local post office wants to distribute a lot of parcels in a city using the minimum of resources such as the number of lorries, the amount of fuel and the number of personnel. The director of a school wants to set up a schedule so that no teacher has classes in parallel while at the same time minimizing their idle times. The participant of a scientific meeting wants to find a fast train connection to the location of a meeting, which does not require too many changes of trains. Optimization problems also occur very frequently in science, such as determining the ground states of physical systems, identifying the native states of proteins, or looking for similarities in ancient languages. Computer scientist have identified certain prototypes of problems behind these real-world and science applications, like finding the shortest paths, the shortest round trips, or satisfying the conditions of Boolean formulas.

For several decades, development in this field took place first of all in applied computer science. The basic task was, and still is, to find algorithms which solve these problems as quickly as possible. Also theoretical computer science, within *algorithmic complexity theory*, considered these problems, in order to classify them according to their "difficulty". A major breakthrough came in 1971, when Cook proved the *NP-completeness* of the *satisfiability problem* (SAT), i. e., he was able to show that SAT is equivalent to all problems from the class NP, which contains the most interesting, i. e., not easily solvable, problems. This breakthrough allowed for a better classification of the problems and has fertilized the field. Since then, literally thousands of problems have been identified as being NP-complete as well. Nevertheless, the notion of NP-completeness is related to the *worst-case* running time of an algorithm. For all NP-complete problems, all known algorithms exhibit a worst-case running time which explodes exponentially with the problem size. However, to study real-world applications, one would also like to develop a notion of *typical-case* complexity. For this purpose, computer scientists started to study the behavior of algorithms defined on suitable parametrized ensembles of random systems. Interestingly, threshold phenomena were found, which were related to a drastic change in the structure of optimal solutions as well as in the typical running time of algorithms. These phenomena show surprising similarities to *phase transitions* in physical systems.

Here statistical physics comes into play, because the consideration of the typical behavior for systems of many particles is the fundamental problem in this field. Many tools have been developed in statistical physics, which allow the study of phase transitions and glassy behavior, which are two basic phenomena occurring in the study of the typical behavior of combinatorial optimization problems. The transfer of knowledge from statistical physics to computer science started in the mid 1980s, when the *simulated annealing method* was invented, which is a fast yet simple algorithm allowing one to find heuristically, very good approximate solutions of combinatorial optimization problems. The idea of this approach is to map the cost function of the optimization problem on the energy function of an equivalent physical system. On the basis of this type of mapping, from the mid 1990s on, concepts and methods from statistical mechanics were applied to study the phase transitions occurring in optimization problems. Since then, many new results have been found, which have not been accessible before by using solely mathematical tools. Also other new algorithms have been developed on the foundation of a physics perspective, e. g., the *survey propagation approach* which allows the treatment of much larger problems, e. g., for SAT, than before. Given the large number of NP-complete problems, the growing range of applications of combinatorial optimization and also the growth of computer power, much progress in this direction can be expected in forthcoming years.

In this book, we introduce concepts and methods for the statistical mechanics of disordered systems. We explain models exhibiting *quenched disorder* such as spin glasses and hard-core gases. We introduce the most important analytical techniques in this field, namely the *replica approach* and the *cavity method*. We also present stochastic algorithms like the Monte-Carlo method or the survey propagation technique, both of which are based on a statistical mechanics viewpoint. In this way the book serves also as an introduction to the statistical mechanics of disordered systems, but with the special point of view of combinatorial optimization.

As in many interdisciplinary contexts, culturally different approaches and languages are present in the different scientific communities. Whereas, e. g., mathematical research work considers mathematical rigor as one of the fundamental building blocks, physical research is usually much more "result oriented". To understand complicated natural phenomena, it is frequently necessary to work with approximations, or to use assumptions which are not immediately justified mathematically.

One aim of this book is helping to bridge the resulting language gap. The book is intended to be accessible for readers with different backgrounds: physicists, who want to enter into this new interdisciplinary area, mathematicians and theoretical computer scientists, who want to understand the methods used by physicists, or applied computer scientists, who want to exploit the insight gained using physics tools to develop new and more efficient algorithms. The level of the book is targeted to graduate students of the different fields, who want to enlarge their horizon beyond the specific borders of their subject of study.

To achieve this, we include various introductory chapters on algorithms, graph theory, complexity theory, and on statistical mechanics. These chapters formulate the minimal basic knowledge for the main part of the book. Readers educated in these fields may want to skip some parts here.

In the central part of the book, we concentrate, for pedagogical reasons and internal coherence, on one specific model, the *vertex cover problem*. We have selected this problem because, on one hand, it shows the main phenomena discussed in the field. On the other hand, the analytical approaches are relatively easy to access compared, e. g., with the most famous combinatorial problem, the satisfiability problem. The details of the latter model are discussed in a specific chapter towards the end of the book. We think that this pedagogical approach enables the reader to understand more easily the vast original literature.

After having discussed in detail the application of statistical mechanics tools to computer-science problems, we also include a chapter discussing the information transfer in the opposite direction, which is historically more strongly developed. Many physical problems can be understood as combinatorial optimization problems. Using various examples, we discuss how computer-science algorithms can be applied to better understand the physical behavior of such systems.

In preparing this book we benefited greatly from many collaborations and discussions with many of our colleagues. We would like to thank Mikko Alava, Simon Alder, Carlo Amoruso, Timo Aspelmeier, Alain Barrat, Wolfgang Barthel, Jürgen Bendisch, Giulio Biroli, Stefan Boettcher, Alfredo Braunstein, Alan Bray, Kurt Borderix, Bernd Burghardt, Ian Campbell, Adam Carter, Loredana Correale, Rodolfo Cuerno, Eytan Domany, Phil Duxbury, Andreas Engel, Martin Feix, Silvio Franz, Ulrich Geyer, Dieter Heermann, Guy Hed, Olaf Herbst, Heinz Horner, Jérôme Houdayer, Michael Jünger, Helmut Katzgraber, Sigismund Kobe, Matthias Koelbel, Werner Krauth, Reiner Kree, Florent Krzakala, Michele Leone, Klaus-Peter Lieb, Frauke Liers, Andreas Linke, Olivier Martin, Alan Middleton, Remi Monasson, Michael Moore, Alejandro Morales, Juan Moreno, Roberto Mulet, Javier Muñoz-García, Uli Nowak, Matthias Otto, Andrea Pagnani, Matteo Palassini, Gerhard Reinelt, Alberto Rosso, Federico Ricci-Tersenghi, Heiko Rieger, Guilhem Semerjian, Eira Seppälä, Dietrich Stauffer, Simon Trebst, Matthias Troyer, Klaus Usadel, Alexei Vazquez, Emmanuel Yewande, Peter Young, Riccardo Zecchina and Annette Zippelius.

This book was prepared at the University of Göttingen and the Institute for Scientific Interchange (ISI), Turin. We would like to acknowledge financial support from the Deutsche Forschungsgemeinschaft (DFG), the VolkswagenStiftung and the European Science Foundation (ESF).

Göttingen and Turin, January 2005

Alexander K. Hartmann and Martin Weigt

1 Introduction

Optimization problems appear in many situations in our daily life. If you, e. g., connect to the web server of a train company or travel agency, you can search connections between far-distant places for optimal travel times, ticket prices or number of changes, etc. In a more general economic context, such problems appear whenever a process has to be arranged in such a way that resources, money, or time are saved. Also in science optimization problems are of central interest, e. g., in physics for understanding the low-temperature behavior of model systems, or in biology for extracting information out of a huge amount of experimental data.

Probably you first encountered optimization in school: In mathematics courses one-dimensional functions over the space of real variables are analyzed, including the determination of minima and maxima. Such problems are in general easily solvable – at least

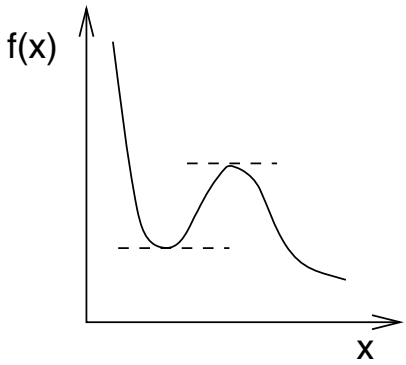


Figure 1.1: Extremal points of a function defined over a one-dimensional continuous space.

on a computer. In this book we are going to discuss so-called combinatorial optimization problems. These are in practice much harder to solve. First, the function to be minimized is in general high-dimensional; second, the variables are of discrete nature, and methods of continuous analysis such as taking derivatives do not work.

In this introductory chapter, we will present two examples which give you a flavor of combinatorial optimization. Furthermore, we will explain why optimization problems are interesting for physicists and how statistical mechanics can contribute to the understanding and solution of them. Finally, we recommend and discuss some basic textbooks related to the field.

1.1 Two examples of combinatorial optimization

Let us start by briefly formalizing the above explanation of combinatorial optimization problems. In general they are defined over some high-dimensional space, we will consider multi-component variables $\underline{\sigma} = (\sigma_1, \ldots, \sigma_n) \in X^n$ with $n \gg 1$. In addition, the problem is characterized by some *cost function* $H : X^n \to \mathbb{R}$ which assigns some cost to each $\underline{\sigma}$. This cost has to be minimized, leading to the following definition:

Definition: *minimization problem*

Given an n-dimensional space X^n and a cost function $H : X^n \to \mathbb{R}$.
The *minimization problem* reads as follows:

$$\textit{Find } \underline{\sigma}^{(0)} \in X^n \textit{, such that } H(\underline{\sigma}) \geq H(\underline{\sigma}^{(0)}) \textit{ for all } \underline{\sigma} \in X^n$$

The vector $\underline{\sigma}^{(0)}$ is called a *solution* of the minimization problem.

Finding a maximum is as easy or as hard as finding a minimum, since $\max H = -\min(-H)$. In addition we speak of *combinatorial* problems if the components of $\underline{\sigma}$ are of *discrete* nature, frequently used examples are $X = \{-1, 1\}$ (Ising spins), $X = \{true, false\}$ (Boolean variables) or $X = \mathbb{Z}$ (integer numbers).

Combinatorial optimization problems can be additionally complicated by the presence of *constraints*. The constraints reduce the size of the search space. At first glance this seems to facilitate the search for an optimal solution. The opposite is, however, frequently the case: Many optimization problems which can be solved efficiently on a computer without constraints, become extremely computer-time consuming if constraints are added.

To illustrate these rather abstract concepts, we will present two examples. The first one is a classical optimization problem in computer science, originating in economics.

Example: Traveling Salesman Problem (TSP)

Consider n towns distributed in a plane, numbered by $1, \ldots, n$. A salesman wants to visit all these towns, and at the end of his travels he wants to come back to his home town. He is confronted with the following minimization task: He has to find the shortest round-tour visiting every town exactly once. The problem is thus described by

$$\begin{aligned} X &= \{1, 2, \ldots, n\} \\ H(\underline{\sigma}) &= \sum_{i=1}^{n} d(\sigma_i, \sigma_{i+1}) \end{aligned} \tag{1.1}$$

where $d(\sigma_i, \sigma_j)$ is the distance between the two towns σ_i and σ_j, and $\sigma_{n+1} \equiv \sigma_1$ are identified with each other. The constraint that every town is visited once and

only once can be realized by constraining the vector $\underline{\sigma}$ to be a permutation of the sequence $(1, 2, \ldots, n)$.

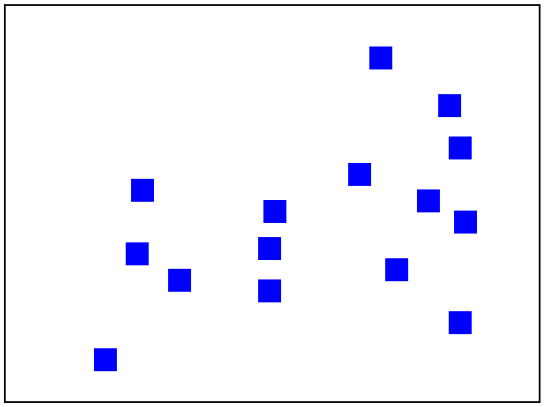


Figure 1.2: 15 cities in a plane.

As an example, 15 towns in a plane are given in Fig. 1.2; and one can try to find the shortest tour. For the general TSP the cities are not necessarily placed in a plane, but an arbitrary distance matrix d is given. □

The TSP is a so-called *hard* optimization problem. In this context, hardness is measured by the time a computer needs to solve this problem numerically, and a problem is considered to be hard if this time grows exponentially (or even faster) with the number n of components of the variable $\underline{\sigma}$. In TSP, one out of $(n-1)!$ round-tours has to be selected, and so far no good general selection criterion exists. Note that TSP has attracted not only computer scientists, but also physicists [1–6].

For an arbitrary algorithm, to describe the dependence between a suitably chosen measure n of the problem size and the running time T, the $\mathcal{O}$ notation is used.

Definition: *$\mathcal{O}$ notation*

Let $T, g : \mathbb{N} \to \mathbb{R}$ be two real-valued functions.
We write $T(n) = \mathcal{O}(g(n))$, iff there exists a positive number $c > 0$, such that $T(n) \leq cg(n)$ is valid for all $n > 0$. We say, $T(n)$ is *of order* at most $g(n)$.

Since constants are ignored when using the $\mathcal{O}$ notation, one speaks of the *asymptotic* running time or *time complexity*. In theoretical computer since, usually on states an upper bound over all possible inputs of size n, i. e., the *worst-case running time*. In this book, we will also study *typical* running times regarding a given ensemble of instances, see Sec. 1.2.

In Table 1.1, orders of running times, which occur typically in the context of algorithms, are presented, accompanied by the resulting values for problem sizes 10, 100, and 1000.

Table 1.1: Growth of functions as a function of input size n.

$T(n)$	$T(10)$	$T(100)$	$T(1000)$
n	10	100	1000
$n \log n$	10	200	3000
n^2	10^2	10^4	10^6
n^3	10^3	10^6	10^9
$n^{\log n}$	10	10^4	10^9
2^n	1024	1.3×10^{30}	1.1×10^{301}
$n!$	3.6×10^6	10^{158}	4×10^{2567}

Usually one considers problems as *easy*, if the running time is bounded by a polynomial, all others are considered as *hard*. The reason can be understood from the table: Even if the polynomial functions may take higher values for small n, asymptotically non-polynomial functions diverge much faster. Let us consider, e. g., the relative performance of two computers, one being twice as fast as the other one. In a linear-time problem, the faster computer is able to solve a problem which is twice as large as the problem solvable in the same time on the slower computer. If the running time grows, however, as 2^n, the faster computer is just able to go from n to $n + 1$ compared with the slower one. We see that for such hard problems, the utility of higher-speed computers is very limited – a substantial increase in the size of solvable problems can only be achieved via the use of better algorithms.

Often it is not immediately obvious whether or not a problem is easy. While finding the shortest round tour in the TSP is hard, finding the shortest path between two given towns (possibly through other cities) is easy, as we will see in Sec. 11.4.

Also in physics, many problems either are, or can be translated into, optimization problems. Examples are the determination of ground states of magnetic systems, the calculation of the structure of a folded protein, the analysis of data, or the study of flux lines in superconductors. This will be illustrated using a classical model in statistical physics.

Example: Ising spin glasses

Spin glasses [7, 8] are amorphous magnetic materials. The atoms in the solid carry microscopically small magnetic moments, which interact via couplings, some are ferromagnetic while others are antiferromagnetic. Due to the amorphous nature of the material, these couplings are disordered, i. e., they do not have any periodic structure. Spin glasses show an interesting frozen low-temperature phase which is, despite ongoing research over more than three decades, still poorly understood. Spin glasses can be modeled in the following way:

The magnetic moments are described by *Ising spins* σ_i which, due to anisotropies of the material, can take only two orientations called *up* and *down*, mathematically formalized by $\sigma_i = \pm 1$. In the simplest model one assumes that these spins are

placed on the sites of a regular lattice and that a spin interacts only with its nearest neighbors. The model is thus described by

$$\begin{aligned} X &= \{-1, 1\} \\ H(\underline{\sigma}) &= -\sum_{<i,j>} J_{ij}\sigma_i\sigma_j \end{aligned} \tag{1.2}$$

where J_{ij} denotes the interaction strength between the spins on sites i and j, and the sum runs over all pairs $\langle i, j\rangle$ of nearest neighbors. The function H measures the total energy of the system, and is called the *Hamiltonian*. Note that the interaction parameters are fixed, they are also called *quenched* variables, whereas the Ising spins are subject to thermal fluctuations and my change their values.

In a *ferromagnet* it is energetically favorable for any two neighboring spins to assume equal orientations $\sigma_i = \sigma_j$, i. e., all J_{ij} are positive, and parallel spins lead to a lower contribution to the total energy than antiparallel ones. So the system tends to be globally oriented in the same way. On the other hand, thermal noise causes spins to fluctuate randomly. At low temperatures T this thermal noise is small, and energetic contributions dominate over random spin fluctuations. The system becomes globally *ordered*. For temperatures higher than some critical temperature T_c, this long-range order becomes destroyed; a *phase transition* occurs at T_c. In Chap. 5, this phenomenon will be discussed in more detail in the context of a short introduction to statistical mechanics.

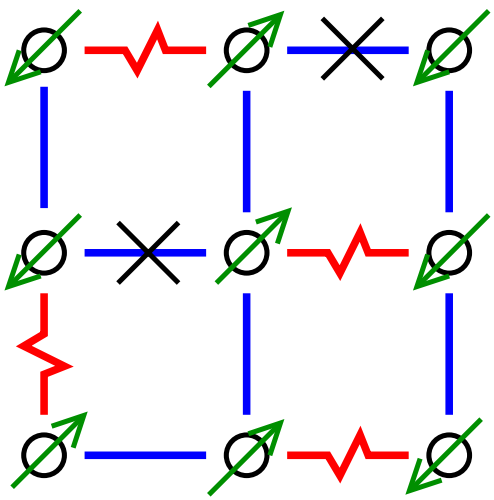


Figure 1.3: Two-dimensional spin glass. Solid lines represent ferromagnetic interactions, while jagged lines correspond to antiferromagnetic interactions. The small arrows represent the spins, adjusted to a ground-state configuration. For all except two interactions (marked by the crosses) the spins are oriented relative to each other in an energetically favorable way. It is not possible to find a state with lower energy.

At temperature $T = 0$ thermal fluctuations are completely absent, and the energy $H(\underline{\sigma})$ of the system assumes its global minimum. The corresponding spin configuration is called a *ground state*. The properties of these configurations are of great interest in physics, they serve as a basis for understanding the low-temperature behavior of physical systems. In the case of a ferromagnet these ground states are, as discussed above, extremely simple; all spins have the same orientation.

This becomes more complicated for *spin glasses* because ferromagnetic interactions ($J_{ij} > 0$) coexist with *antiferromagnetic* ones ($J_{ij} < 0$). Two spins connected by such an antiferromagnetic coupling prefer energetically to have opposite orientations. As already noted, spin glasses are amorphous materials, therefore ferromagnetic and antiferromagnetic interactions are distributed randomly on the bonds of the lattice. Consequently, it is not obvious how the ground state configurations of $H(\underline{\sigma})$ appear, and determining the minimum energy becomes a non-trivial minimization problem.

Figure 1.3 shows a small two-dimensional spin glass and one of its ground states. For this type of system usually many different ground states are feasible for each realization of the disorder. One says, the ground state is *degenerate*. Algorithms for calculating degenerate spin-glass ground states are explained in Chap. 11. As we will explain there, this calculation is easy (polynomial running time) for two-dimensional systems, but it becomes hard (exponential running time) in three dimensions. □

1.2 Why study combinatorial optimization using statistical physics?

The interdisciplinary exchange between computer science and statistical physics goes in both directions:

On one hand, many problems in daily life and in many scientific disciplines can be formulated as optimization problems. Hence, progress in the theory of combinatorial optimization, and in particular the development of new and more efficient algorithms influences many fields of science, technology and economy. In this obvious sense, computer science contributes to the understanding of physical systems. We will give a short introduction on the application of optimization algorithms for physics problems in Chap. 11.

On the other hand, physics can also help to shed light onto some basic, yet unsolved, questions in computer science. In statistical mechanics, many analytical and numerical methods have been developed in order to understand the macroscopic thermodynamic behavior of models starting from some microscopic description of a system via its Hamiltonian. These methods can be reinterpreted from the point of view of optimization theory. Problems, which are originally formulated as purely combinatorial tasks, can be equivalently rewritten as physical

models, by identifying the cost function with a Hamiltonian. Applying statistical mechanics tools at a temperature close to $T = 0$, may thus unveil many properties of the original problem and its cost-function minima; this will be the main focus of this book. In this way we will obtain insight into the intrinsic reasons for the hardness of these problems. For example, we will see that the hardness of many optimization problem is closely related to the *glassiness* of the corresponding physical system. Using this alternative physical approach, many interesting results have already been obtained, which have not been found before by applying traditional methods from mathematics and computer science.

Furthermore, traditional computer science defines the hardness according to a *worst-case* scenario. People dealing with practical applications are, however, more interested in *typical* instances of an optimization task rather than looking for the hardest possible instances. For this reason, suitably parametrized random ensembles of instances of problems have been introduced over recent years. In this context, it was observed that in some regions of the ensemble space instances are *typically* easy to solve, i. e., in a polynomially increasing running time, while in other regions instances are found to be typically hard. This change in behavior resembles the phase transitions observed in physical systems, like spin glasses or other disordered materials. Once more it is tempting to exploit the long experience of statistical physics with phase transitions in order to understand this behavior.

Example: Phase transitions in the TSP

As an example, we consider again the TSP. We study the random ensemble, where a plane of area size $A = L_x \times L_y$ is given. Each random instance consists of n cities, which are randomly placed in the plane, with $(x_i, y_i) \in [0, L_x] \times [0, L_y]$ denoting the position of city i. All positions are equally probable. The distances between pairs of cities are just Euclidean distances, i. e., $d(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

For each random instance of the problem, we ask the question:

Is the shortest round trip through all cities shorter than a given length l?

To answer this question for a given instance, one can use a *branch-and-bound* algorithm. For an introduction to this type of algorithm, see Sec. 6.3. Here, the algorithm basically enumerates all possible permutations of n cities stepwise, by selecting one city after the other. Some efficient heuristic is used, which determines the order in which the cities are tried, e. g., one starts the construction of the round trip, by first taking the two cities which are closest to each other, for details see Ref. [9]. The algorithm furthermore maintains a lower bound $l_{\min}$ of the tour length, determined by the cities selected so far. If $l_{\min} > l$, then the algorithm does not have to pursue permutations containing the same order of the cities selected so far, and it can search in other regions of the permutation space. On the other hand, if a full round trip with $H(\underline{\sigma}) < l$ has been found, one knows that the answer to the question above is "yes", and the algorithm can stop.

In Fig. 1.4, the probability p that a tour of length smaller than l exists, is shown [9] as a function of the rescaled length $\Phi = l/\sqrt{nA}$. One observes that there is a strong

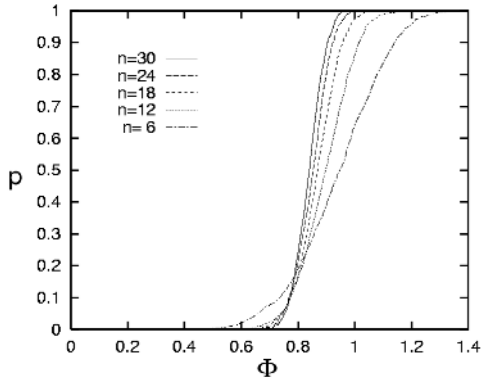


Figure 1.4: Probability p that the minimum tour of n cities in plane of area A is shorter than $\Phi = l/\sqrt{nA}$, for different number n of cities. Reprinted from Ref. [9] with permission from Elsevier.

increase in p when increasing l and that the curves for different sizes cross close to $\Phi = 0.78$. Hence, there seems to be a fundamental change in the nature of the question close to $\Phi = 0.78$, which resembles a *phase transition* in physical systems, see Sec. 5.1.

One can show [9] that all probability curves for different sizes n fall on top of each other, when p is plotted as a function of $r = (l/\sqrt{nA} - 0.78)n^{2/3}$. In this sense, the problem exhibits a certain type of universality, independent of the actual number of cities.

In Fig. 1.5 the running time of the TSP algorithm, measured by the number of times a position of some city is assigned, when creating permutations, is shown as a function of the parameter r. One observes that, close to the point, where exactly half of the instances have a shorter (minimum tour) length than l, the running time is maximal. This means that close to the phase transition, the problem is hardest to solve. It is a well known phenomenon in physics that, for the computer simulations of phase diagrams, the running time is maximal close to phase transitions. It is one of the main purposes of this book, to understand how this coincidence arises for combinatorial optimization problems.

Finally, we mention that it is easy to understand why the running time is small for large as well as for small values of Φ (i. e., l):

- For very small values of l, even the two closest cities have a larger distance than the given value of l, hence the algorithm can stop immediately after one step.
- For very large values of l, even the first permutation of the cities has a total distance smaller than l, i. e., the algorithm stops after $n - 1$ steps.

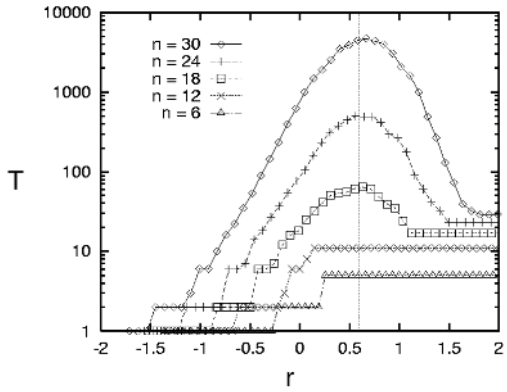


Figure 1.5: Running time for branch-and-bound algorithm as a function of $r = (l/\sqrt{nA} - 0,78)n^{2/3}$, for different number n of cities. The vertical line indicates where half of the instances have a shorter tour length than l. Reprinted from Ref. [9] with permission from Elsevier.

□

A better understanding of problems and algorithms may help to approach the solution of the ultimate task: to find fast algorithms such that larger instances can be handled efficiently. This is the main part of the work of computer scientists working in algorithmic design. Also in this field, physics has recently proved its strength. For example the *simulated annealing* technique originates in physics, see Chap. 2. The basic idea is to mimic the experimental cooling of a system to find the low-temperature, i. e., low-energy, behavior. This technique has the advantage of being widely applicable in very different fields. The disadvantage is that it does not guarantee finding the global optimum. For many problems in practice, it is, however, sufficient to find some local optima, which are close to the global one. Much more recently, the deep insight into the structural properties of the solutions of several hard optimization problems gained by statistical-mechanics analysis, has led to the proposal of a new class of statistical inference algorithms, called *survey propagation*. These algorithms are discussed in detail in Chaps 9 and 10.

1.3 Textbooks

In the following list, some useful basic textbooks are suggested.

- T. H. Cormen, S. Clifford, C. E. Leiserson, R. L. Rivest: *Introduction to Algorithms*, MIT Press, 2001
  This book is considered as the standard introduction to algorithms and data structures and provides a good foundation in the field which proves to be useful for theoretical studies and especially for practical applications and implementations of algorithms at all levels.

- A. V. Aho, J. E. Hopcroft, J. D. Ullman: *The design and analysis of computer algorithms*, Addison-Wesley, 1974
  This book merges the field of practical applications of algorithms and theoretical studies in this field.

- M. R. Garey and D. S. Johnson: *Computers and intractability*, Freeman, New York, 1979
  This book is a traditional standard text book on complexity theory. It concentrates on the part of theoretical computer science related to hard problems. The basic classes of problems are defined, many fundamental problems are explained, and their relationship is proved. In addition, the book contains a huge list of combinatorial problems which may serve as a source of inspiration for further research.

- C. H. Papadimitriou and K. Steiglitz: *Combinatorial Optimization*, Prentice-Hall, 1982
  This book gives a good introduction to the field of combinatorial optimization. All relevant basic problems and algorithms are explained. It exists in an economic paperback edition.

- A. K. Hartmann and H. Rieger: *Optimization Algorithms in Physics*, Wiley-VCH, Berlin, 2001
  This text book shows how optimization algorithms can be applied to many problems in physics. It explains the transformations needed to convert physical problems into optimization problems, and presents the algorithms needed to solve these problems.

- M. Mézard, G. Parisi, and M. A. Virasoro: *Spin glass theory and beyond*, World Scientific, Singapore, 1987.
  This book gives an introduction to the statistical-mechanics theory of spin glasses, together with reprints of the most important papers. It discusses also first applications of spin-glass methods to non-physical problems, including neural networks and combinatorial optimization.

- K. H. Fisher and J. A. Hertz: *Spin Glasses*, Cambridge University Press, 1991
  This book introduces the methods of statistical physics needed to study phase transitions in optimization problems. In this text, all techniques are applied to spin glasses, for which the methods were originally developed.

Bibliography

[1] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, Science **220**, 671 (1983).

[2] M. Mézard and G. Parisi, J. Physique **47**, 1285 (1986).

[3] N. J. Cerf, J. Boutet de Monvel, O. Bohigas, O. C. Martin, and A. G. Percus, Journal de Physique I **7**, 117 (1997).

[4] A. G. Percus and O. C. Martin, J. Stat. Phys. **94**, 739 (1999).

[5] A. Chakraborti and B. K. Chakrabarti, Eur. Phys. J. B **16**, 677 (2000).

[6] D. S. Dean, D. Lancaster, and S. N. Majumdar, arXiv preprint `cond-mat/0411111`.

[7] M. Mézard, G. Parisi, and M. A. Virasoro, *Spin Glass Theory and Beyond* (World Scientific, Singapore, 1987).

[8] K. H. Fisher and J. A. Hertz, *Spin Glasses* (Cambridge University Press, 1991).

[9] I. P. Gent and T. Walsh, Artif. Intell. **88**, 349 (1996).

2 Algorithms

This chapter gives a short introduction to algorithms. We first present a notation called *pidgin Algol* which will be used throughout this book to describe algorithms. Next, we introduce some basic principles of algorithms frequently encountered later on when studying optimization techniques: iteration, recursion, divide-and-conquer, dynamic programming and backtracking. Since there are many specialized textbooks in this field [1–3] we will demonstrate these fundamental techniques by presenting only a few simple examples.

2.1 Pidgin Algol

The algorithms will not be presented using a specific programming language. Instead, we will use a *notation* for algorithms called *pidgin Algol*, which resembles modern high-level languages like Algol, Pascal or C. But unlike any conventional programming language, variables of an arbitrary type are allowed, e. g., they can represent numbers, strings, lists, sets or graphs. It is not necessary to declare variables and there is no strict syntax.

For the definition of pidgin Algol, we assume that the reader is familiar with at least one high-level language and that terms like *variable*, *expression*, *condition* and *label* are clear. A pidgin Algol program is a sequence of *statements* between a **begin** and an **end**, or, in other words a *compound* statement (see below). It is composed of the following building blocks:

1. *Assignment*
   variable := *expression*

   A value is assigned to a variable. Examples: $a := 5 * b + c$, $A := \{a_1, \ldots, a_n\}$

   Also more complex and informal structures are allowed, like
   let z be the first element of the queue Q

2. *Condition*
   if *condition* **then** *statement 1*
   else *statement 2*

   The **else** clause is optional. If the condition is true, statement 1 is executed, otherwise statement 2 is executed (if it exists).

   Example: **if** *money*$>$100 **then** *restaurant* := 1 **else** *restaurant* := 0

3. *Cases*
   case: *condition 1*
   statement1_A;
   statement1_B;
   ...
   case: *condition 2*
   statement2_A;
   statement2_B;
   ...
   case: *condition 3*
   statement3_A;
   statement3_B;
   ...
   ...
   end cases

   This statement is useful if many different case can occur, thus making a sequence of **if** statements too complex. If condition 1 is true, then the first block of statements is executed (here no **begin** ... **end** is necessary). If condition 2 is true, then the second block of statements is executed, etc.

4. *While loop*
   while *condition* **do** *statement*

   The statement is performed as long as the condition is true.

   Example: **while** *counter* < 200 **do** *counter* := *counter*+1;

5. *For loop*
   for *list* **do** *statement*

   The statement is executed for all parameters in the list. Examples:

   for $i := 1, 2, \ldots, n$ **do** *sum* := *sum*$+i$

   for all elements q of queue Q **do** *waits*[q] := *waits*[q]+1

6. *Goto statement*
   a) *label: statement*
   b) **goto** *label*

   When the execution of an algorithm reaches a goto statement the execution is continued at the statement which carries the corresponding label.

7. *Compound statement*
   begin
   statement 1;
   statement 2;
   ...
   statement n;
   end

The compound statement is used to convert a sequence of statements into one statement. It is useful, e. g., if a for-loop should be executed for a body of several statements.

Example:
for $i := 1, 2, \ldots, n$ **do**
begin
$a := a + i$;
$b := b + i * i$;
$c := c + i * i * i$;
end

For brevity, sometimes a compound statement is written as a list of statements in one line, without the **begin** and **end** keywords.

8. *Procedures*
procedure *procedure-name* (*list of parameters*)
begin
statements
return *expression*
end

The **return** statement is optional. Note that we use the **return** statement also inside algorithms to return the final result. A procedure is used to define a new name for a collection of statements. A procedure can be invoked by writing: *procedure-name* (*arguments*)

Example:

procedure minimum (x, y)
begin
if x>y **then return** y
else return x
end

9. *Comments*
comment *text*

Comments are used to explain parts of an algorithm, i. e., to aid in its understanding. Sometimes a comment is given at the right end of a line without the **comment** keyword.

10. *Miscellaneous statements*: practically any text which is self-explanatory is allowed. Examples:

Calculate determinant D of matrix M
Calculate average waiting time for queue Q

As a first example we present a simple heuristic for the TSP problem, which we have introduced in Sec. 1.1. This method constructs a tour which is quite short, but it does not guarantee to find the optimum. The basic idea is to start at a randomly chosen city. Then iteratively the

city which has the shortest distance from the present city, i. e., its *nearest neighbor*, is chosen from the set of cities which have not yet been visited. The array v will be used to indicate which cities already belong to the tour. In σ_i the ith city visited is stored. Please remember that $d(i, j)$ denotes the distance between cities i and j and n is the number of cities.

algorithm TSP-nearest-neighbor($n, \{d(i, j)\}$)
begin
  for $i := 1, 2, \ldots, n$ **do**
    $v[i] := 0$;
  σ_1 := one arbitrarily chosen city;
  $v[\sigma_1] := 1$;
  for $i := 2, 3, \ldots, n$ **do**
  begin
    min $:= \infty$;
    for all j with $v[j] = 0$ **do**
      if $d(\sigma_{i-1}, j) <$ min **then**
        min $:= d(\sigma_{i-1}, j)$; $\sigma_i := j$;
    $v[\sigma_i] := 1$;
  end
end

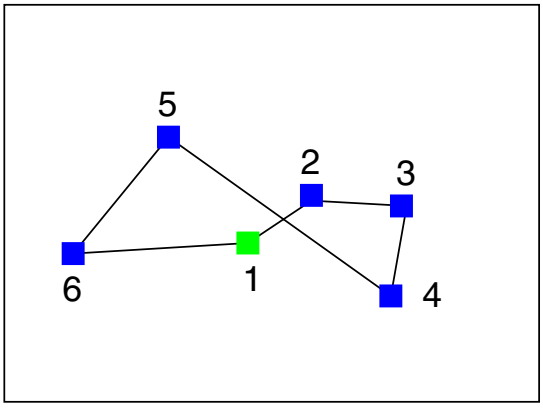


Figure 2.1: Example of where the heuristic fails to find the shortest round trip.

Please note that the length of the tour constructed in this way depends on the city where the tour starts and that this city is randomly chosen. As already mentioned, the algorithm does not guarantee to find the shortest tour. This can be seen in Fig. 2.1, where the algorithm is applied to a sample. In the resulting tour, two edges ($(1, 2)$ and $(4, 5)$) cross, which shows that a shorter round trip exists. One can replace the two crossing edges by $(1, 4)$ and $(2, 5)$.

2.2 Iteration and recursion

If a program has to perform many similar tasks, this can be expressed as an iteration, also called a loop, e. g., with the **while**-statement from pidgin Algol. Sometimes it is more convenient to use the concept of *recursion*, especially if the quantity to be calculated has a recursive definition. One speaks of recursion if an algorithm/procedures calls itself (maybe indirectly through other algorithms/procedures). As a simple example we present an algorithm for the calculation of the factorial $n!$ of a natural number $n > 0$. Its recursive definition is given by:

$$n! = \begin{cases} 1 & \text{if } n = 0 \text{ or } n = 1 \\ n \times (n-1)! & \text{else} \end{cases} \tag{2.1}$$

This definition can be translated directly into an algorithm:

algorithm factorial(n)
begin
  if $n \leq 1$ **then**
    return 1
  else
    return n*factorial($n-1$)
end

In the first line the test is whether $n \leq 1$ instead of testing for $n = 0$ or $n = 1$. Therefore, it is guaranteed that the algorithm returns something on all inputs.

For $n > 1$, during the execution of factorial(n), a sequence of nested calls of the algorithm is created up to the point where the algorithm is called with argument 1. The call to factorial(n) begins before and is finished after all other calls to factorial(i) with $i < n$. The hierarchy in Fig. 2.2 shows the calls for the calculation of factorial(4).

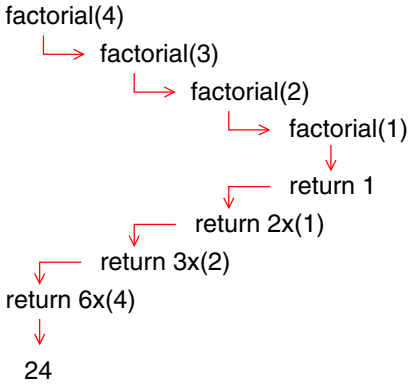


Figure 2.2: Hierarchy of recursive calls for calculation of factorial(4).

Every recursive algorithm can be rewritten as a sequential algorithm, containing no calls to itself. Instead loops are used. Usually, sequential versions are faster by some constant factor but harder to understand, at least when the algorithm is more complicated than in the present example. The sequential version for the calculation of the factorial reads as follows:

algorithm factorial2(n)
begin
  $t := 1$; **comment** this is a counter
  $f := 1$; **comment** here the result is stored
  while $t \leq n$ **do**
  begin
    $f := f * t$;
    $t := t + 1$;
  end
  return f;
end

The sequential factorial algorithm contains one loop which is executed n times. Thus, the algorithm runs in $\mathcal{O}(n)$ steps, remember that the $\mathcal{O}$ notation was introduced in Sec. 1.1. For the recursive variant the time complexity is not so obvious. For the analysis of recursive algorithms, one has to write a *recurrence* equation for the execution time. For $n = 1$, the factorial algorithm takes a constant time $T(1)$. For $n > 1$ the algorithm takes the time $T(n-1)$ for the execution of factorial($n - 1$) plus another constant time for the multiplication. Here and in the following, let C be the maximum of all occurring constants. Then the running time is bounded from above by $\tilde{T}(n)$ given by

$$\tilde{T}(n) = \begin{cases} C & \text{for } n = 1 \\ C + \tilde{T}(n-1) & \text{for } n > 1\,. \end{cases} \tag{2.2}$$

One can verify easily that $\tilde{T}(n) = Cn$ is the solution of the recurrence, i. e., both recursive and sequential algorithms have the same asymptotic time complexities. There are many examples where a recursive algorithm is asymptotically faster than a straightforward sequential solution. An example will be given in the following section, see also [1].

2.3 Divide-and-conquer

The basic idea of *divide-and-conquer* is to divide a problem into smaller subproblems, solve the subproblems and then combine the solutions of the subproblems to form the final solution. Recursive calls of the algorithm are usually applied here as well.

As an example we consider the multiplication of binary numbers $x = x_1x_2 \ldots x_n$, $y = y_1y_2 \ldots y_n$ having n bits $x_i, y_i \in \{0, 1\}$, for convenience we assume $n = 2^k$. The direct multiplication runs over all bits x_i of x and adds $y2^{n-i}$ if $x_i = 1$. Since the shift (multiplication

by 2^{n-i}) and the addition operations need a time linear in n, the full multiplication takes a running time $\mathcal{O}(n^2)$. We will now present a $\mathcal{O}(n^{\log_2(3)}) \approx \mathcal{O}(n^{1.58})$ running time algorithm using the divide-and-conquer approach.

The basic idea is to divide each n-bit number into two $n/2$-bit numbers:

$$\begin{aligned} x &= (x_1 \dots x_{n/2})(x_{n/2+1} \dots x_n) = (a)(b) = a2^{n/2} + b \\ y &= (y_1 \dots y_{n/2})(y_{n/2+1} \dots y_n) = (c)(d) = c2^{n/2} + d\,. \end{aligned}$$

Then the multiplication reads as follows:

$$\begin{aligned} xy &= (a2^{n/2} + b)(c2^{n/2} + d) = ac2^n + (ad + bc)2^{n/2} + bd \\ &= ac2^n + ([a + b][c + d] - ac - bd)2^{n/2} + bd\,. \end{aligned}$$

Please note that in the second line, we use only three *different* multiplications ac, bd and $[a + b][c + d]$ instead of four as in the first line. We will see that this is crucial for obtaining an asymptotically fast algorithm. Our idea is to use only multiplications of $n/2$-bit numbers. The only remaining difficulty is that $[a+b]$ and $[c+d]$ may be $(n/2+1)$-bit numbers due to a carry arising from the addition. The solution is to split the two sums $s = a + b$ and $t = c + d$ into the leading bits s_1, t_1 and the remaining numbers s_R,t_R which are each $n/2$ bits long. The resulting algorithm reads as follows, assuming that short multiplications of, at most, $n_{\min}$ bits can be performed quickly by the computer:

algorithm mult($x = (a)(b), y = (c)(d)$)
begin
  $n :=$ number of bits of x, y
  if $n \leq n_{\min}$ **then**
    return(xy);
  else
  begin
    $(s_1)(s_R) := (s_1)(s_2 \dots s_{n/2+1}) := s := a + b$;
    $(t_1)(t_R) := (t_1)(t_2 \dots t_{n/2+1}) := t := c + d$;
    $u := s_1 t_1 2^n + (s_1 t_R + s_R t_1)2^{n/2}$+mult($s_R, t_R$); **comment** $= (a + b)(c + d)$
    $v :=$ mult(a, c);
    $w :=$ mult(b, d);
    return($v2^n + (u - v - w)2^{n/2} + w$);
  end
end

Additions and shifts of bits (i. e., multiplications by 2^n or $2^{n/2}$) can be performed in a time linear in the number n of bits. Thus, the algorithm replaces an n-bit multiplication by three $n/2$-bit multiplications and several $\mathcal{O}(n)$ operations. To obtain the running time, we again write down a recurrence equation. Assuming $n_{\min} = 1$, the running time is bounded from

above by $T(n)$ given as the solution of

$$T(n) = \begin{cases} k & \text{for } n = 1 \\ 3T(n/2) + kn & \text{for } n > 1\,. \end{cases} \tag{2.3}$$

The solution to this equation is $T(n) = 3kn^{\log_2(3)} - 2kn$. This can be easily shown by induction (please note that $3 = 2^{\log_2(3)}$):

$$\begin{aligned} n = 1: \\ T(1) &= k \\ n \to 2n: \\ T(2n) &= 3T(n) + k2n \\ &= 3(3kn^{\log_2(3)} - 2kn) + k2n \\ &= 3k(2n)^{\log_2(3)} - 2k(2n)\,. \end{aligned}$$

Note that the general recurrence equation

$$T(n) = \begin{cases} k & \text{for } n = 1 \\ aT(n/c) + kn & \text{for } n > 1 \end{cases} \tag{2.4}$$

is, for $a > c$, solved by $T(n) = \mathcal{O}(n^{\log_c(a)})$. Hence, when using the direct replacement of an n-bit multiplication by four $n/2$-bit multiplications (i. e., $a = 4$, $c = 2$) one obtains again an algorithm with running time $\mathcal{O}(n^2)$. Therefore, it is important to have three multiplications instead of four in order to obtain a calculation which is faster than the naive approach.

2.4 Dynamic programming

Another problem where the application of divide-and-conquer and recursion seems quite natural is the calculation of *Fibonacci numbers* $\mathrm{fib}(n)$. Their definition is as follows:

$$\mathrm{fib}(n) = \begin{cases} 1 & (n = 1) \\ 1 & (n = 2) \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2) & (n > 2)\,. \end{cases} \tag{2.5}$$

Thus, e. g., $\mathrm{fib}(4) = \mathrm{fib}(3) + \mathrm{fib}(2) = (\mathrm{fib}(2) + \mathrm{fib}(1)) + \mathrm{fib}(2) = 3$, $\mathrm{fib}(5) = \mathrm{fib}(4) + \mathrm{fib}(3) = 3 + 2 = 5$. The functions grow very rapidly: $\mathrm{fib}(10) = 55$, $\mathrm{fib}(20) = 6765$, $\mathrm{fib}(30) = 83204$, $\mathrm{fib}(40) > 10^8$. Let us assume that this definition is translated directly into a recursive algorithm. Then a call to $\mathrm{fib}(n)$ would call $\mathrm{fib}(n-1)$ and $\mathrm{fib}(n-2)$. The recursive call of $\mathrm{fib}(n-1)$ would call *again* $\mathrm{fib}(n-2)$ and $\mathrm{fib}(n-3)$ [which is also called from the two calls of $\mathrm{fib}(n-2)$, etc.]. The total number of calls increases rapidly with n, even more than $\mathrm{fib}(n)$ itself increases with n. In Fig. 2.3, the top of a hierarchy of calls is

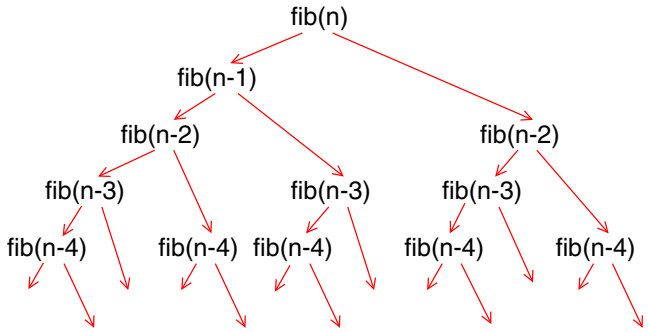


Figure 2.3: Hierarchy of calls for $\mathrm{fib}(n)$.

shown. Obviously, every call to fib with a specific argument is performed frequently, which is definitely a waste of time.

Instead, one can apply the principle of *dynamic programming*. The basic idea is to start with small problems, solve them and store them for later use. Then one proceeds with larger problems by using divide-and-conquer. If, for the solution of a larger problem, a smaller one is necessary, it is already available. Therefore, no direct recursive calls are needed. As a consequence, the performance increases drastically. The divide-and-conquer algorithm for the Fibonacci numbers reads as follows, the array $f[]$ is used to store the results:

algorithm fib-dynamic(n)
begin
  if $n < 3$ **then**
    return 1;
  $f[1] := 1$;
  $f[2] := 1$;
  for $i := 3, 4, \ldots, n$ **do**
    $f[i] := f[i-1] + f[i-2]$
  return $f[n]$;
end

Since the algorithm contains just one loop, it runs in $\mathcal{O}(n)$ time. Finally, we should point out that there is an explicit formula which allows a direct calculation of the Fibonacci numbers:

$$\mathrm{fib}(n) = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right). \tag{2.6}$$

2.5 Backtracking

The last basic programming principle which is presented here is *backtracking*. This method is applied when there is no direct way of computing a solution. This is typical for many optimization problems. Remember the simple TSP algorithm, which constructs a feasible tour but does not guarantee to find the optimum. In order to improve the method, one has to try some (sub-)solutions, discard them if they turn out not to be good enough and try some other (sub-)solutions. This is the basic principle of backtracking.

In the following we will present a backtracking algorithm for the solution of the N-queens problem.

Example: N-queens problem

N queens are to be placed on an $N \times N$ chess board in such a way that no queen checks against any other queen.

This means that, in each row, in each column and in each diagonal at most one queen is placed. □

A naive solution of the algorithms works by enumerating all possible configurations of N queens and checking, for each configuration, whether any queen checks against another queen. By restricting the algorithm to place at most one queen per column, there are N^N possible configurations. This is a very strongly increasing running time, which can be decreased by backtracking.

The idea of backtracking is to place one queen after the other. One stops placing further queens if a non-valid configuration is already obtained at an intermediate stage. Then one goes one step back, removes the queen which was placed at the step before, places it elsewhere, if possible, and continues again.

In the algorithm, we use an array Q_i which stores the position of the queen in column i. If $Q_i = 0$, no queen has been placed in that column. For simplicity, it is assumed that N and the array Q_i are global variables. Initially all Q_i are zero.

The algorithm starts in the last column and places a queen. Next a queen is placed in the second last column by a recursive call and so forth. If all columns are filled, a valid configuration has been found. If at any stage it is not possible to place any further queen in a given column then the backtracking-step is performed: the recursive call finishes, the queen which was set in the recursion-step before is removed. Then it is placed elsewhere and the algorithm proceeds again. The argument n denotes the column where the next queen is to be placed. Hence, initially the algorithm is called with $n = N$.

```
algorithm queens(n)
begin
    if n = 0 then
        print array Q_1, ..., Q_N; problem solved;
    for i := 1, ..., N do
    begin
        Q_n := i    comment place queen in column n and row i
        if Queen n does not check against any other then
            queens(n − 1);
    end
    Q_i := 0;
    return
end
```

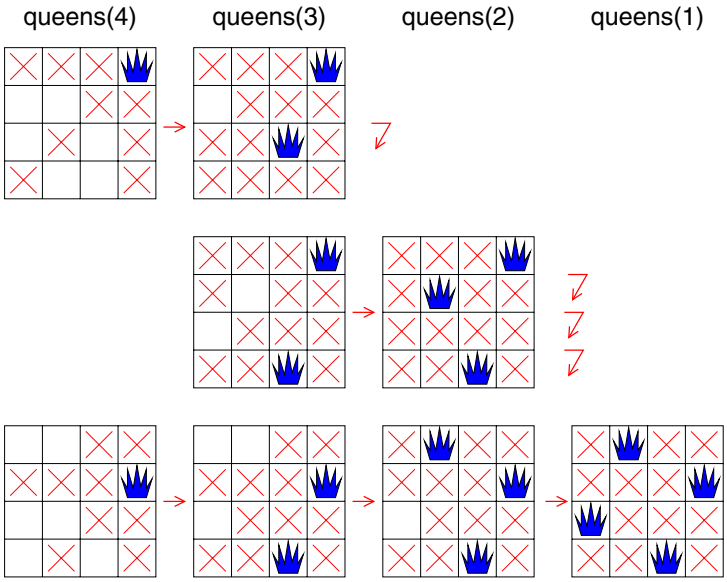


Figure 2.4: How the algorithm solves the 4-queens problem.

In Fig. 2.4 the way in which the algorithm solves the problem for $N = 4$ is shown. It starts with a queen in column 4 and row 1. Then queens(3) is called. The positions where no queen is allowed are marked with a cross. For the third column no queens in row 1 and row 2 are allowed. Thus, a queen is placed in row 3 and queens(2) is called. In the second column it is not possible to place a queen. Hence, the call to queens(2) finishes. The queen in column 3 is placed one row below, i. e., row 4 (second line in Fig. 2.4). Then, by calling queens(2), a queen is placed in row 2 and queens(1) is called. Now, no queen can be placed in the first column, hence queen(1) returns. Since there was only one possible position in column 2, the queen is removed and the also the call queens(2) finishes. Now, both possible positions for the

queen in column 3 have been tried. Therefore, the call for queens(3) finishes as well and we are back at queens(4). Now, the queen in the last column is placed in the second row (third line in Fig. 2.4). From here it is straightforward to place queens in all columns and the algorithm succeeds.

Although this algorithm avoids many "dead ends" it still has an exponential time complexity as a function of N. This is quite common in many hard optimization problems. More sophisticated algorithms, which we will encounter later, try to reduce the number of configurations visited even further.

Bibliography

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, Reading (MA) 1974)

[2] R. Sedgewick, *Algorithms in C*, (Addison-Wesley, Reading (MA) 1990).

[3] T. H. Cormen, S. Clifford, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, (MIT Press, 2001).

3 Introduction to graphs

The next three sections give a short introduction to graph theory and graph algorithms. The first one deals with the basic definitions and concepts, and introduces some graph problems. The second one is dedicated to some fundamental graph algorithms. In the third one, we will discuss random graphs, which will be of fundamental importance throughout this course.

Let us begin by mentioning some books related to graph theory. All of them go well beyond everything we will need concerning graphs:

- Gary Chartrand, *Introductory graph theory*, Dover Publ. Inc., New York, 1985.
  This little paperback contains a nice, easy-to-read introduction to graph theory. Every chapter is based on "real-world" examples, which are mapped to graph problems. It is a good book for everyone who wishes to know more about graphs without working through a difficult mathematical book.

- Béla Bollobás, *Modern Graph Theory*, Springer, New York 1998.
  Written by one of the leading graph theorists, this book covers the first and third section (and much more). It is very good and complete, but mathematically quite difficult.

- Robert Sedgewick, *Algorithms in C: Graph Algorithms*, Addison Wesley, Boston 2002.
  This book collects many graph algorithms, including extensive descriptions and proofs of their correctness.

3.1 Basic concepts and graph problems

3.1.1 The bridges of Königsberg and Eulerian graphs

The earliest use of graph theoretical methods probably goes back to the 18th century. At this time, there were seven bridges crossing the river Pregel in the town of Königsberg. The folks had long amused themselves with the following problem: Is it possible to walk through the town using every bridge just once, and returning home at the end? The problem was solved by Leonhardt Euler (1707–1783) in 1736 by mapping it to a graph problem and solving it for arbitrary graphs [1], i. e., for arbitrary towns, city maps, etc. In the case of Königsberg, he had to draw the slightly disappointing consequence that no such round-walk existed.

Figure 3.1 shows the river Pregel, Königsberg was situated on both sides and both islands. The seven bridges are also shown. The mapping to a graph is given below. Every river side, island and bridge is assigned a *vertex*, drawn as a circle, two vertices are connected by an *edge*, drawn as a line, if they are also physically connected. To give a more precise description, we have to introduce the basic graph-theoretical terminology which is summarized in the definitions below.

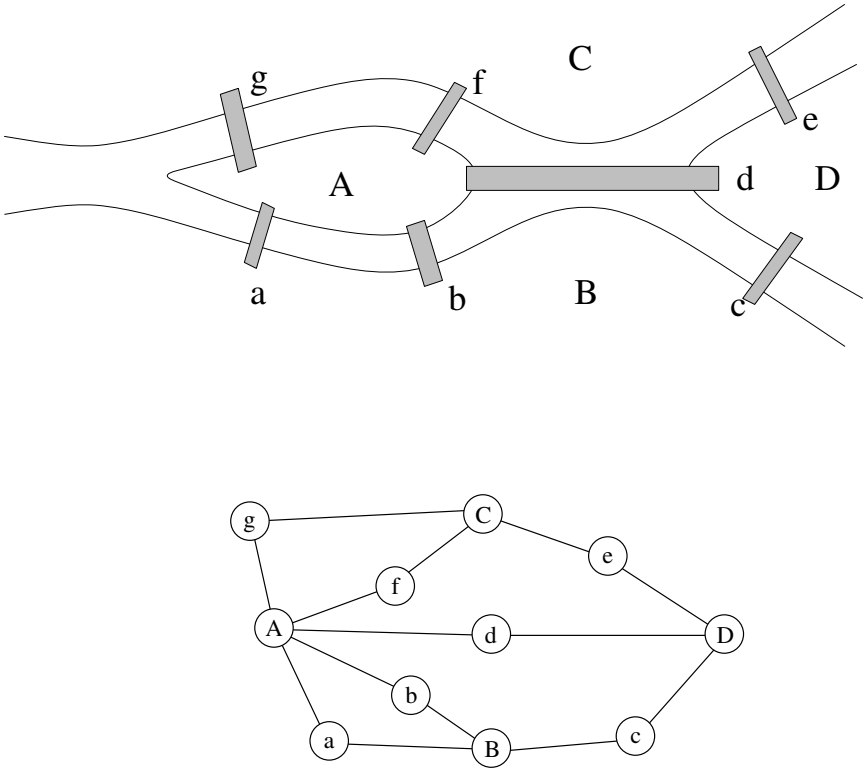


Figure 3.1: The bridges of Königsberg and its graph representation. Vertices are denoted by circles, edges by lines.

Basic definitions:

- An *(undirected) graph* $G = (V, E)$ is given by its *vertices* $i \in V$ and its undirected *edges* $\{i, j\} \in E \subset V^{(2)}$. Note that both $\{i, j\}$ and $\{j, i\}$ denote the same edge.
- The *order* $N = |V|$ counts the number of vertices.
- The *size* $M = |E|$ counts the number of edges.
- Two vertices $i, j \in V$ are *adjacent / neighboring* if $\{i, j\} \in E$.
- The edge $\{i, j\}$ is *incident* to its end vertices i and j.

- The *degree* $\deg(i)$ of vertex i equals the number of adjacent vertices. Vertices of zero degree are called *isolated*.
- A graph $G' = (V', E')$ is a *subgraph* of G if $V' \subset V,\ E' \subset E$.
- A graph $G' = (V', E')$ is an *induced subgraph* of G if $V' \subset V$ and $E' = E \cap (V')^{(2)}$, i. e., E' contains all edges from E connecting two vertices in V'.
- A subgraph $G' = (V', E')$ is a *path* of G if it has the form $V' = \{i_0, i_1, \ldots, i_l\}$, $E' = \{\{i_0, i_1\}, \{i_1, i_2\}, \ldots, \{i_{l-1}, i_l\}\}$. The *length* of the path is $l = |E'|$. i_0 and i_l are called *end points*. The path goes from i_0 to i_l and vice versa. One says i_0 *and* i_l *are connected by the path*. Note that, within a path, each vertex (possibly except for the end points) is "visited" only once.
- A path with $i_0 = i_l$, i. e., a *closed path*, is called a *cycle*.
- A sequence of edges $\{i_0, i_1\},\ \{i_1, i_2\}, \ldots, \{i_{l-1}, i_l\}$ is called a *walk*. Within a walk some vertices or edges may occur several times.
- A walk with pairwise distinct edges is called a *trail*. Hence a trail is also a subgraph of G.
- A *circuit* is trail with coinciding end points, i. e., a *closed* trail. (NB: cycles are circuits, but not vice versa, because a circuit may pass through several times through the same vertex).
- The graph G is *connected* if all pairs i, j of vertices are connected by paths.
- The graph $G' = (V', E')$ is a *connected component* of G if it is a connected, induced subgraph of G, and there are no edges in E connecting vertices of V' with those in $V \setminus V'$.
- The *complement graph* $G^C = (V, E^C)$ has edge set $E^C = V^{(2)} \setminus E = \{\{i, j\} \mid \{i, j\} \notin E\}$. It is thus obtained from G by connecting all vertex pairs by an edge, which are not adjacent in G and disconnecting all vertex pairs, which are adjacent in G.
- A *weighted graph* $G = (V, E, \omega)$ is a graph with edge weights $\omega : E \to \mathbb{R}$.

Example: Graphs

We consider the graph shown in Fig. 3.1. It can be written as $G = (V, E)$ with

$$\begin{aligned} V &= \{A, B, C, D, a, b, c, d, e, f, g\} \\ E &= \{\{A, a\}, \{A, f\}, \{A, d\}, \{A, f\}, \{A, g\}, \{B, a\}, \{B, b\}, \{B, c\}, \\ &\quad \{C, e\}, \{C, f\}, \{C, g\}, \{D, c\}, \{D, d\}, \{D, e\}, \}\,. \end{aligned}$$

Hence, the graphs has $|V| = 11$ vertices and $|E| = 14$ edges. Since $\{D, e\} \in E$, the vertices D and d are adjacent. Vertex d has degree $\deg(d) = 2$, while vertex A has degree 5.

For example, $G' = (V', E')$ with $V' = \{A, g, C, e, D\}$ and $E' = \{\{A, g\}, \{g, C\}, \{C, e\}, \{e, D\}, \}$ is a path from A to D. G is connected, because all vertices are connected by paths to all other vertices. The sequence of edges $\{B, c\}, \{c, D\}, \{D, c\}$

is a walk, but it does not correspond to a path, because some vertices are visited twice. The sequence of edges $\{A, b\}$, $\{b, B\}$, $\{B, a\}$, $\{a, A\}$, $\{A, g\}$, $\{g, C\}$, $\{C, f\}$, $\{f, A\}$ is a trail, in particular it is a circuit. □

Going back to the problem of the people from Königsberg, formulated in graph-theoretical language, they were confronted with the following question:

EULERIAN CIRCUIT: Given a graph, is there a circuit using every *edge* exactly once?

The amazing point about Euler's proof is the following: The existence of a Eulerian circuit – which obviously is a global object – can be decided looking to purely local properties, in this case to the vertex degrees.

Theorem: *A connected graph $G = (V, E)$ is Eulerian (has an Eulerian cycle) iff all vertex degrees are even.*

Proof:

($\rightarrow$) This direction is trivial. Following the Eulerian circuit, every vertex which is entered is also left, every time using previously unvisited edges. All degrees are consequently even.

($\leftarrow$) The other direction is a bit harder. We will proof it by induction on the graph size M, for arbitrary graphs having only even vertex degrees.

The theorem is obviously true for $M = 0$ (one isolated vertex) and $M = 3$ (triangle) which are the simplest graphs with even degrees.

Now we take any $M > 0$, and we assume the theorem to be true for all graphs of size smaller than M. We will show that the theorem is also true for a connected graph G of size M having only even degrees.

Because of $M > 0$, the graph G is non-trivial, because of the even degrees it contains vertices of degree at least 2. Then G contains a cycle, which can be seen in the following way: Start in any vertex and walk along edges. Whenever you enter a new vertex, there is at least one unused edge which you can use to exit the vertex again. At a certain moment you enter a vertex already seen (at most after M steps), the part of the walk starting there is a cycle.

Every cycle is also a circuit. Consider now a circuit $C = (G', E')$ of maximal size $|E'|$. If C is a Eulerian circuit in G, everything is OK. If not, we have $|E'| < |E|$, and the subgraph $H = (V, E \setminus E')$ has at least one non-trivial connected component H'. A circuit has even degrees, thus H and all its connected components have even degrees. Since H' has size $< M$, it has an Eulerian circuit which can be added to C, violating the maximality of C. Thus, C has to be an Eulerian circuit, and the proof is complete. QED

Going back to Königsberg, we see that there are vertices of odd degrees. No Eulerian circuit can exist, and the inhabitants have either to skip bridges or to cross some twice if they walk through their town.

In the above definitions for undirected graphs, edges have no orientation. This is different for *directed* graphs (also called *digraphs*). Most definitions for directed graphs are the same as for undirected graphs. Here we list some differences and additional definitions.

- A *directed graph* $G = (V, E)$ is similar to an undirected graph, except that the edges (also called *arcs* in this case) $(i, j) \subset V \times V$ are ordered pairs of vertices.
- The *outdegree* $d_{\text{out}}(i)$ is the number of outgoing edges (i, j).
- The *indegree* $d_{\text{in}}(i)$ is the number of incoming edges (j, i).
- A directed *path* is defined similarly to a path for undirected graphs, except that all edges must have the same "forward" orientation along the path. Formally, a path form i_0 to i_l is a subgraph $G' = (V', E')$ with $V' = \{i_0, i_1, \ldots, i_l\}$, $E' = \{(i_0, i_1), (i_1, i_2), \ldots, (i_{l-1}, i_l)\}$.
- A directed graph $G = (V, E)$ is called *strongly connected* if for all pairs of vertices i, j, there exists a directed path in G from i to j and a path from j to i. A *strongly connected component* (SCC) of a directed graph is a strongly connected subgraph of maximal size, i. e., it cannot be extended by adding vertices/edges while still being strongly connected.

3.1.2 Hamiltonian graphs

Imagine you have to organize a diplomatic dinner. The main problem is placing the N ambassadors around a round table: There are some countries which are strong enemies, and during the dinner there is a risk that the diplomats will not behave very diplomatically. You therefore prefer to place only those ambassadors in neighboring seats who belong to countries which have peaceful relations.

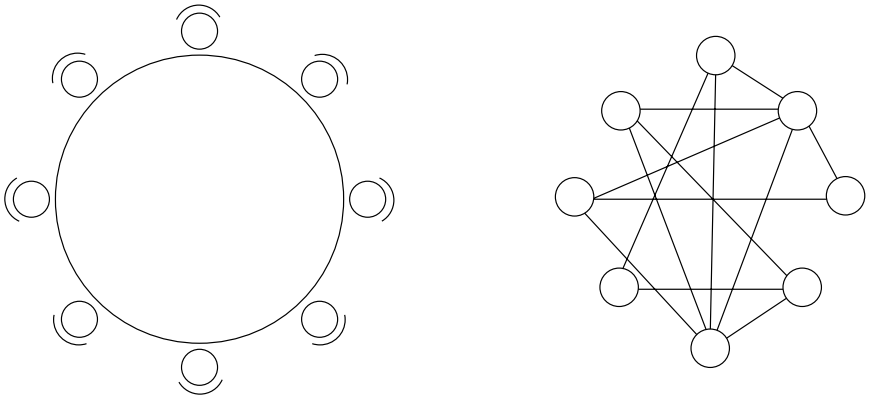


Figure 3.2: Left: The round table where the diplomats have to be placed. Right: Graph of relations, only ambassadors with positive relations are connected and can be neighbors at the table.

Again, this problem can be mapped to a graph-theoretical question. The corresponding graph is the "graph of good relations". The vertices are identified with the ambassadors, and any two having good relations, i. e., potential neighbors, are connected by an edge, see Fig. 3.2. The problem now reduces to finding a cycle of length N:

***Hamiltonian Cycle*:** (HC) Is there a cycle through a given graph such that every *vertex* is visited exactly once?

At first, this problem seems to be similar to the search for an Eulerian circuit, but it is not. It belongs to the NP-complete problems (see Chap. 4) and is, in general, not decidable from local considerations. An exception is given in the case of highly connected graphs:

Theorem: *If $G = (V, E)$ is a graph of order $N \geq 3$ such that $\deg(i) \geq N/2$ for all vertices in V, then G is Hamiltonian (i. e., it has a Hamiltonian cycle).*

We do not give a proof, we just state that the theorem is not sufficient to say anything about our specific sample given in Fig. 3.2. There are vertices of degree 2 and 3, which are smaller than $N/2 = 4$.

The search for Hamiltonian cycles is closely related to the already mentioned traveling-salesman problem (TSP), which is defined on a *weighted* graph. The *total weight of a subgraph* is the sum of the weights of all its edges. In TSP we are looking for the Hamiltonian cycle of minimum weight.

3.1.3 Minimum spanning trees

Let us come to the next problem. This time you have to plan a railway network in a country having no trains but many towns which are to be connected. The country has financial problems and asks you to design the cheapest possible network, providing you with the information of which pairs of towns can be directly connected, and at what cost – i. e. they give you a weighted graph. You must find a connected subgraph containing each town to enable travel from one town to another. It is also clear that loops increase the total cost. The most expensive edge in a loop can be deleted without disconnecting some towns from others. But how can one find the globally optimal solution?

Before answering this, we have to go through some definitions:

Definition: *tree, forest*

A *tree* is a connected cycle-free graph.
A *forest* is a graph which has only trees as connected components, see, e. g., Fig. 3.3.

Some easy-to-prove properties of trees are given here:

Theorem: *A tree of order N has size $M = N - 1$.*

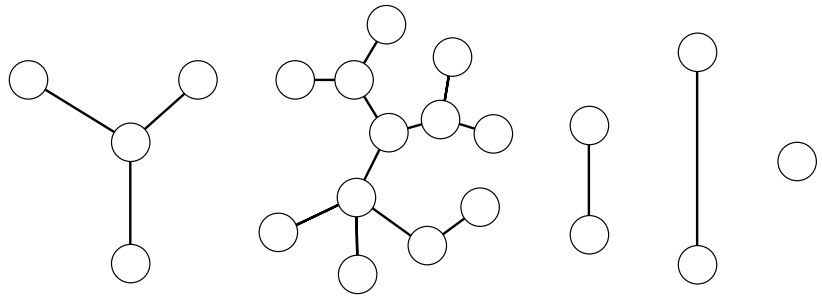


Figure 3.3: A forest is made of trees.

Proof:

Start with one isolated vertex, add edges. Since there are no cycles, every edge adds a new vertex. QED

Corollary: *Every tree of order $N \geq 2$ contains at least two vertices of degree 1 (leaves).*

Proof:

Assume the opposite, i. e., that there are at least $N - 1$ vertices of degree at least 2. We thus find

$$\sum_{i \in V} \deg(i) \geq 2N - 1$$

On the other hand, we have

$$\sum_{i \in V} \deg(i) = 2M$$

since every edge is counted twice. This produces a contradiction to the theorem. QED

To approach our railway-network problem, we need two further definitions.

Definition: *(Minimum) spanning tree/forest*

- Let $G = (V, E)$ be a connected graph of order N. A *spanning tree* is a cycle-free subgraph of order N having maximum size $N - 1$, i. e., the spanning tree is still connected.
- If $G = (V, E, \omega)$ is weighted, a spanning tree of minimum weight is called an *minimum spanning tree* (or *economic spanning tree*).
- For a general graph, a *(minimum) spanning forest* is the union of all (minimum) spanning trees for its different connected components.

Due to the maximum size condition of a spanning tree, the connected components of a graph and of its spanning forest are the same. Spanning trees can be constructed using algorithms which calculate connected components, see Sec. 3.2.1. Minimum spanning trees obviously fulfil all requirements to our railway network, and they can be constructed in a very simple way using Prim's algorithm, which is presented in Sec. 3.2.3.

3.1.4 Edge covers and vertex covers

Now we introduce edge and vertex covers. Both problems are similar to each other, but they are fundamentally different with respect to how easily they can be solved algorithmically. This we have already seen for Eulerian circuits and Hamiltonian cycles. The vertex-cover problem will serve as the prototype example in this book. All concepts, methods and analytical techniques will be explained using the vertex-cover problem. We begin here with the basic definitions, but additional definitions, related to vertex-cover algorithms, are given in Sec. 6.1.

For a graph $G = (V, E)$, an *edge cover* is a subset $E' \subset E$ of edges such that each vertex is contained in at least one edge $e \in E'$. Each graph which has no isolated vertices has an edge cover, since in that case E itself is an edge cover. A *minimum edge cover* is an edge cover of minimum cardinality $|E^{'}|$. In Fig. 3.4 a graph and a minimum edge cover are shown. A fast algorithm which constructs a minimum edge cover can be found in Ref. [2].

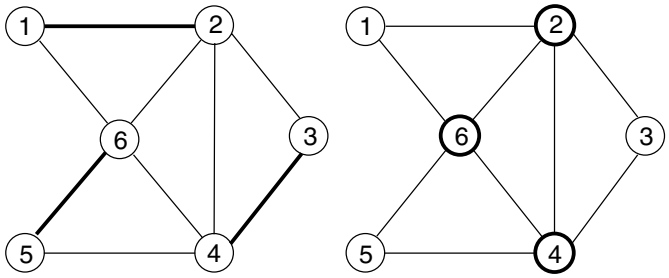


Figure 3.4: A graph and a minimum edge cover (left) and a minimum vertex cover (right). The edges/vertices belonging to the cover are shown in bold.

The definition of a *vertex cover* is very similar. It is a subset $V' \subset V$ of vertices such that each edge $e = \{i, j\} \in E$ contains at least one vertex out of V', i. e., $i \in V'$ or $j \in V'$. Note that V itself is always a vertex cover. A *minimum vertex cover* is an vertex cover of minimum cardinality $|V^{'}|$.

Vertex covers are closely related to *independent sets* and *cliques*. An independent set of a graph $G = (V, E)$ is a subset $I \subset V$ of vertices, such that for all elements $i, j \in I$, there is no edge $\{i, j\} \in E$. A clique is a subset $Q \subset V$ of vertices, such that for all elements $i, j \in Q$ there is an edge $\{i, j\} \in E$.

Theorem: *For a given graph $G = (V, E)$ and a subset $V' \subset V$ the following three statements are equivalent.*

(A) *V' is a vertex cover of G.*

(B) *$V \setminus V'$ is an independent set of G.*

(C) *$V \setminus V'$ is a clique of the complement graph G^C (see definition on page 27).*

Proof:

(A $\rightarrow$ B)
Let V' be a vertex cover, and $i, j \in V \setminus V'$. We assume that there is an edge $\{i, j\} \in E$. Since $i, j \notin V'$, this is an edge with both vertices not in V', and V' is not a vertex cover. This is a contradiction! Hence, there cannot be an edge $\{i, j\} \in E$, and $V \setminus V'$ is an independent set.

(B $\rightarrow$ C)
Let $V \setminus V'$ be an independent set, and $i, j \in V \setminus V'$. By definition, there is no edge $\{i, j\} \in E$, and so there is an edge $\{i, j\} \in E^C$. Therefore, $V \setminus V'$ is a clique of G^C.

(C $\rightarrow$ A)
Let $V \setminus V'$ be a clique of G^C, and $\{i, j\} \in E$. This means $\{i, j\} \notin E^C$ by definition of G^C. Thus, we have $i \notin V \setminus V'$ or $j \notin V \setminus V'$ because $V \setminus V'$ is a clique. Hence, $i \in V'$ or $j \in V'$. By definition of vertex cover, V' is a vertex cover. QED

The *minimum vertex cover* is a vertex cover of minimum cardinality. From the theorem above, for a minimum vertex cover V', $V \setminus V'$ is a maximum independent set of G and a maximum clique of G^C. In Fig. 3.4, a graph together with its minimum vertex cover is displayed. Related to the minimization problem is the following decision problem, for given integer K:

***Vertex Cover (VC)*:** Does a given graph G have a vertex cover V' with $|V^{'}| \leq K$?

In Chap. 4 we will see that VC is a so-called NP-complete problem, which means in particular that no fast algorithm for solving this problem is known – and not even expected to be able to be constructed. This proposition and its relation to statistical physics will be the main subject of this book.

3.1.5 The graph coloring problem

The last problem which we discuss in this section is a scheduling problem. Imagine you have to organize a conference with many sessions which, in principle, can take place in parallel. Some people, however, want to participate in more than one session. We are looking for a perfect schedule in the sense that every one can attend all the sessions he wants to, but the total conference time is minimum.

The mapping to an undirected graph is the following: The sessions are considered as vertices. Two of them are connected by an edge whenever there is a person who wants to attend both. We now try to *color* the vertices in such a way that no adjacent vertices carry the same color.

Therefore we can organize all those sessions in parallel, which have the same color, since there is nobody who wants to attend both corresponding sessions. The optimal schedule requires the minimum possible number of colors.

The problem is an analog to the coloring of geographic maps, where countries correspond to vertices and common borders to edges. Again we do not want to color neighboring countries using the same color. The main difference is the structure of the graph. The "geographic" one is two-dimensional, without crossing edges – the so-called *planarity* of the graph makes the problem easy. The first graph is more general, and therefore more complicated. Again, the problem becomes NP-complete.

Let us be more precise:

Definition: *q-coloring, chromatic number, q-core*

- A *q-coloring* of $G = (V, E)$ is a mapping $c : V \to \{1, \ldots, q\}$ such that $c(i) \neq c(j)$ for all edges $\{i, j\} \in E$.
- The minimum number of needed colors is called the *chromatic number* of G.
- The maximal subgraph, where the minimum degree over all vertices is at least q, is called the *q-core*.

The smallest graph which is not q-colorable is the *complete graph* K_{q+1} of order $q + 1$ and size $q(q + 1)/2$, i. e., all pairs of vertices are connected. Complete graphs are also called *cliques*. For K_3 this is a triangle, for K_4 a tetrahedron.

Before analyzing the q-coloring problem, a linear-time algorithm for constructing the q-core is introduced. The idea of the algorithm is easy, it removes all vertices of degree smaller than q, together with their incident edges. The reduced graph may have new vertices of degree smaller than q, which are removed recursively, until a subgraph of minimum degree at least q is obtained. For obvious reasons the q-core is obtained: No vertex out of the q-core can be removed by the algorithm, so the final state is at least as large as the q-core. By the maximality of the latter, both have to be equal. The input to the algorithm presented below is the initial graph $G = (V, E)$, the output is its q-core:

algorithm core(V, E, q)
begin
  if $\min\{\deg(i) | i \in V\} \geq q$ **then**
    return $G = (V, E)$;
  else do
    begin
      $U := \{i \in V | \deg(i) < q\}$;
      $V := V \setminus U$;
      $E := E \cap V^{(2)}$;
      return core(V, E, q);
    end
end

Theorem: *A graph is q-colorable if and only if its q-core is q-colorable.*

Proof:

($\rightarrow$) A coloring of the full graph is obviously a coloring of every subgraph.

($\leftarrow$) Construct the q-core by the core-algorithm, but keep track of the order in which vertices are removed. Now take any q-coloring of the core. Reconstruct the full graph by adding the removed vertices in inverse order. Directly after adding, the vertices have degree smaller than q. Their neighbors use less than q colors, i. e., the added vertex can be colored. In this way, every coloring of the q-core can be recursively extended to a coloring of the full graph. QED

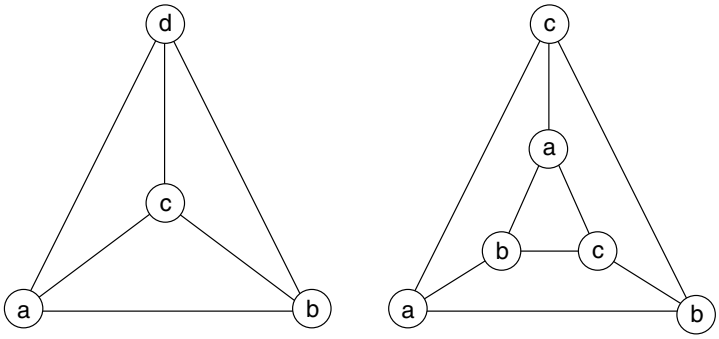


Figure 3.5: The left graph is the tetrahedron K_4 which is not 3-colorable, the letters indicate the different colors of a 4-coloring. The right graph also has fixed degree 3, but is colorable with three colors.

This shows that the hard part of q-coloring a graph consists in coloring its q-core, i. e., the existence of a non-trivial q-core is necessary, but not sufficient for q-uncolorability, see, e. g., the examples given in Fig. 3.5. This also leads to

Corollary: *Every graph having at most q vertices of degree at least q is q-colorable.*

In the beginning of this section we have also considered the coloring of geographic maps – or *planar graphs*, i. e., graphs which can be drawn in a plane without crossing edges. For these graphs, the chromatic number can be found easily, using the following famous theorem:

Theorem: *Every planar graph is 4-colorable.*

This had already been conjectured in 1852, but the final solution was only given in 1976 by Appel and Haken [3]. Their proof is, however, quite unusual: Appel and Haken "reduced" the original question to more than 2000 cases according to the graph structure. These cases were finally colored numerically, using about 1200 hours of computational time. Many mathematicians were quite dissatisfied by this proof, but up to now nobody has come up with a "hand-written" one.

On the other hand, it is easy to see that 3 colors are in general not sufficient to color a planar graph. The simplest example is the 4-clique K_4, which is planar, as can be seen in Fig. 3.5.

3.1.6 Matchings

Given a graph $G = (V, E)$, a matching $M \subset E$ is a subset of edges, such that no two edges in M are incident to the same vertex [4, 5], i. e., for all vertices $i \in V$ we have $i \in e$ for at most one edge $e \in M$. An edge contained in a given matching is called *matched*, other edges are *free*. A vertex belonging to an edge $e \in M$ is *covered* (or *matched*) others are *M-exposed* (or *exposed* or *free*). If $e = \{i, j\}$ is matched, then i and j are called *mates*.

A matching M of maximum cardinality is called *maximum-cardinality matching*. A matching is *perfect* if it leaves no exposed vertices, hence it is automatically a maximum-cardinality matching. On the other hand, not every maximum-cardinality matching is perfect.

The vertex set of a *bipartite graph* can be subdivided into two disjoint sets of vertices, say U and W, such that edges in the graph $\{i, j\}$ only connect vertices in U to vertices in W, *with no edges* internal to U or W. Note that nearest-neighbor hypercubic lattices are bipartite, while the triangular and face-centered-cubic lattices are *not* bipartite. Owing to the fact that all cycles on bipartite graphs have an even number of edges, matching on bipartite graphs is considerably easier than matching on general graphs. In addition, maximum-cardinality matching on bipartite graphs can be easily related to the maximum-flow problem [6–8], see Sec. 11.5.

Example: Matching

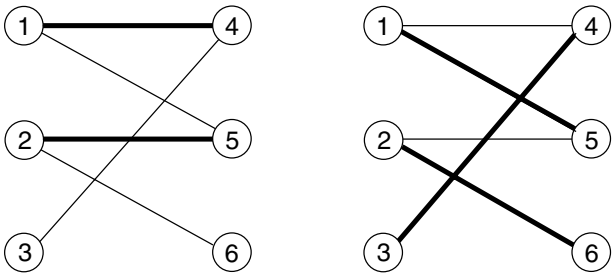


Figure 3.6: Example graph for matching, see text.

In Fig. 3.6 a sample graph is shown. Please note that the graph is bipartite with vertex sets $U = \{1, 2, 3\}$ and $W = \{4, 5, 6\}$. Edges contained in the matching are indicated by thick lines. The matching shown in the left half is $M = \{\{1, 4\}, \{2, 5\}\}$. This means, e. g., edge $\{1, 4\}$ is matched, while edge $\{3, 4\}$ is free. Vertices 1,2,4 and 5 are covered, while vertices 3 and 6 are exposed.

In the right half of the figure, a perfect matching $M = \{\{1, 5\}, \{2, 6\}, \{3, 4\}\}$ is shown, i. e., there are no exposed vertices. □

Having a weighted graph $G = (V, E, w)$, we consider also *weighted matchings*, with the weight of a matching given by the sum of the weights of all matched edges. M is a *maximum-weight matching* if its total weight assumes a maximum with respect to all possible matchings. For perfect matchings, there is a simple mapping between maximum-weight matchings and minimum-weight matchings, namely: let $\tilde{w}_{ij} = W_{\max} - w_{ij}$, where w_{ij} is the weight of edge $\{i, j\}$ and $W_{\max} > \max_{\{i,j\}}(w_{ij})$. A maximum perfect matching on $\tilde{w}_{ij}$ is then a minimum perfect matching on w_{ij}.

A good historical introduction to matching problems, the origins of which may be traced to the beginnings of combinatorics, may be found in Lovász and Plummer [5]. Matching is directly related to statistical mechanics because the partition function for the two-dimensional Ising model on the square lattice can be found by counting *dimer coverings* (= perfect matchings) [5]. This is a graph *enumeration* problem rather than an *optimization* problem as considered here. As a general rule, graph enumeration problems are *harder* than graph-optimization problems.

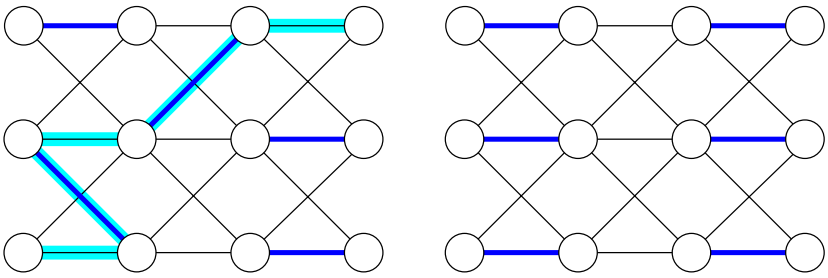


Figure 3.7: (Left): A bipartite graph with a matching. An alternating path (shaded) starts and ends at exposed vertices, and alternates between unmatched (thin) and matched (thick) edges. (Right): Interchanging matched and unmatched edges along an alternating path increases the cardinality of the matching by one. This is called *augmentation* and is the basic tool for maximum-matching algorithms. In this example, the augmentation yields a maximum-cardinality matching, even a perfect matching.

The ground state calculation of planar spin glasses is equivalent to the minimum-weight perfect matching problem on a general graph, see Sec. 11.7. Maximum/minimum (perfect) matching problems on general graphs are algorithmically more complicated than on bipartite graphs, see, e. g., Refs [9, 10], but still they can be solved in a running time increasing only polynomially with the system size. These matching algorithms rely on finding *alternating paths*, which are paths along which the edges are alternately matched and unmatched, see Fig. 3.7.

3.2 Basic graph algorithms

In this section, we present basic graph algorithms, which are related to finding the connected components of a graph. First, we explain two different strategies, the depth-first search and

the breadth-first search. In addition to yielding the connected components, they construct also spanning tress. Then we explain how the strongly connected components of directed graphs can be obtained by modifying the depth-first search. At the end we show, for the case of weighted graphs, that minimum-weight spanning tress can be found using a simple algorithm by Prim. Another fundamental type of graph algorithm is the shortest-path algorithm, which we explain later on in Sec. 11.4.

3.2.1 Depth-first and breadth-first search

For a given graph $G = (V, E)$, we want to determine its connected components. There are two basic search strategies, *depth-first search* (DFS) and *breadth-first search* (BFS), which are closely related.

The main work of the first algorithm is done within the following procedure depth-first(), which starts at a given vertex i and visits all vertices which are connected to i. The main idea is to perform recursive calls of depth-first() for all neighbors of i which have not been visited at that moment. The array *comp*[] is used to keep track of the process. If *comp*$[i] = 0$ vertex i has not been visited yet. Otherwise it contains the number (id) t of the component currently explored. The value of t is also passed as a parameter, while the array *comp*[] is passed as a reference, i. e., it behaves like a global variable and all changes performed to $comp[]$ in a call to the procedure persist after the call is finished.

procedure depth-first(G, i, [ref.] *comp*, t)
begin
  comp$[i]$:=t;
  for all neighbors j of i **do**
    if *comp*$[j]$ = 0 **then**
      depth-first(G, j, *comp*, t);
end

To obtain all connected components, one has to call depth-first() for all vertices which have not been visited yet. This is done by the following algorithm.

algorithm components(G)
begin
  initialize *comp*$[i] := 0$ for all $i \in V$;
  $t := 1$;
  while there is a vertex i with *comp*$[i]$=0 **do**
    depth-first(G, i, *comp*, t); $t := t + 1$;
end

For any vertex i, when the procedure calls itself recursively for a neighbor j, it follows the edge $\{i, j\}$. Since no vertex is visited twice, those edges form a spanning tree of each connected component. We call those edges *tree edges* and the other edges are known as *back edges*.

The algorithm tries to go as far as possible within a graph, before visiting further neighbors of the vertex in which the search started. It is for this reason that the procedure received its name. Thus, the spanning tree is called the *depth-first spanning tree* and the collection of the spanning trees of all connected components is a depth-first spanning forest. In the next section, we will use the depth-first spanning trees to find the strongly connected components in a directed graph.

Example: Depth-first spanning tree

As an example, we consider the graph shown on the left of Fig. 3.8. We assume that the algorithm first calls depth-first() for vertex 1, hence $comp[1] := 1$ is set.

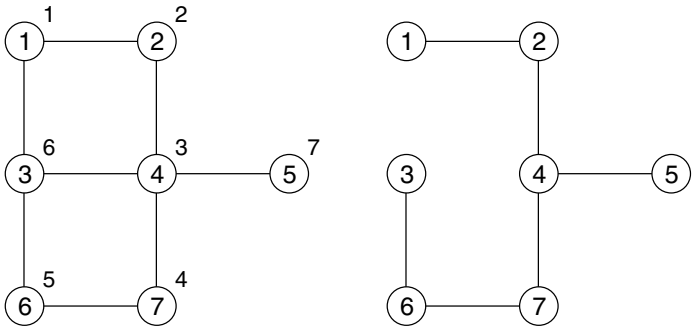


Figure 3.8: A sample graph (left). The number close to the vertices indicates a possible order in which the vertices are visited during a depth-first search. On the right, the resulting depth-first spanning tree is shown.

We assume that vertex 2 is the first neighbor of vertex 1 encountered in the **for** loop. Hence depth-first is called for vertex 2, where $comp[2] := 1$ is set. Here, in the **for** loop, vertex 1 may be encountered first, but it has already been visited ($comp[1] = 1$), hence nothing happens. In the next iteration the second neighbor, vertex 4 is encountered. Now depth-first() is called for vertex 4 and there $comp[4] := 1$ is assigned. We assume that vertex 7 is the first vertex encountered in the loop over all neighbors of vertex 4. Therefore, depth-first($G, 7, comp, 1$) is called next, leading to two more recursive calls of depth-first(). The full recursive hierarchy of calls appears as follows (we show only the parameter for vertex i):

depth-first(1)
  depth-first(2)
    depth-first(4)
      depth-first(7)
        depth-first(6)
          depth-first(3)
      depth-first(5)

The deepest point in the recursion is reached for vertex 3. All its neighbors, vertices 1, 4 and 6 have already been visited, hence the procedure returns here without

further calls. At this point only vertex 5 has not been visited, hence it is visited when iterating over the other neighbors of vertex 4. The order in which the vertices are visited is indicated in Fig. 3.8 and the resulting depth-first spanning tree is also shown. □

During a run of the algorithm, for each vertex all neighbors are accessed to test whether it has already been visited. Thus each edge is touched twice, which results in a time complexity of $\mathcal{O}(|E|)$.

The depth-first search can also be adapted for directed graphs. In this case, being at vertex i, one follows only edges (i, j), i. e., which point in a forward direction. This means that the resulting depth-first spanning trees may look different. Even the vertices contained in a tree may differ, depending on the order in which the vertices are visited in the outer loop (as in components() for the undirected case), as we will see in the example below. This means that components are not well defined for directed graphs, because they depend on the order in which the vertices are treated. Instead, one is interested in calculating the strongly connected components, which are well defined. The corresponding algorithm is presented in the next section. There we will need the the *reverse topological order*,also called *postorder*. This is just the order in which the treatment of a vertex i *finishes* during the calculation of the depth-first spanning tree, i. e., the order in which the calls to the procedure, with i as argument, terminate.

The algorithm reads as follows, it is only slightly modified with respect to the undirected case. The postorder is stored in array $post[]$, which is passed as reference, i. e., all modifications to the values are also effective outside the procedure. We also need a counter c, also passed by reference, which is used to establish the postorder. The array $tree$, passed also by reference, keeps track of which vertices have already been visited and stores the identification numbers of the trees in which the vertices are contained, corresponding to the array $comp$ used for the undirected case.

procedure depth-first-directed(G, i, [ref.] $tree$, t, [ref.] $post$, [ref.] c)
begin
  $tree[i] := t$;
  for all j with $(i, j) \in E$ **do**
    if *tree*[j] = 0 **then**
      depth-first-directed(G, j, $tree$, t, $post$, c);
  $post[i] := c$;
  $c := c + 1$;
end

The main subroutine, finding all connected depth-first spanning trees and establishing the reverse topological order, reads as follows.

algorithm postorder(G)
begin
  initialize *tree*$[i] := 0$ for all $i \in V$;
  $t := 1$;
  $c := 1$;
  while there is a vertex i with $tree[i]$=0 **do**
    depth-first-directed(G, i, $tree$, t, $post$, c); t:=t+1;
end

The vertices, for which depth-first-directed() is called inside the loop in postorder(), i. e., the vertices where the construction of a tree starts, are called *roots* of the depth-first spanning trees. They are the vertices which receive the highest $post$ numbers for each tree. Vertices of other trees, which are visited earlier receive lower $post$ numbers, vertices of trees being visited later will receive higher $post$ numbers than all vertices of the current tree.

Example: Depth-first spanning tree of a directed graph

We consider the graph shown in Fig. 3.9.

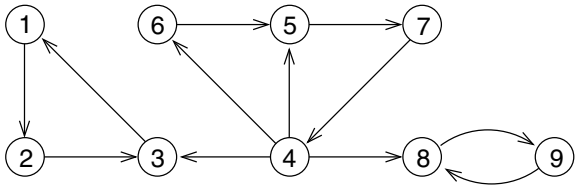


Figure 3.9: A sample directed graph.

We assume that in postorder() vertex 1 is first considered. Vertex 1 has one neighbor, hence depth-first-directed() will be called for vertex 2, in which the procedure is called for vertex 3. Vertex 3 has one neighbor in the forward direction, vertex 1, which has already been visited, hence no call is performed here. This means that the procedure finishes for vertex 3, and $post[3] := 1$ is assigned. Since all neighbors of vertex 2 have been visited, this call also finishes and $post[2] = 2$ is assigned. Next the same thing happens for vertex 1, resulting in $post[1] = 3$. Hence, the algorithm returns to the top level, to postorder().

Now, we assume that depth-first-directed is called for vertex 4, and that its neighbors are processed in the order 3,5,6,8. Vertex 3 has already been visited. The call of depth-first-directed() for vertex 5 leads to a call of the procedure for its unvisited neighbor vertex 7. The call for vertex 8 itself calls the procedure for vertex 9. In total the following calling hierarchy is obtained, only the passed values for the vertex i are shown.

depth-first-directed(1)
  depth-first-directed(2)
    depth-first-directed(3)
depth-first-directed(4)
  depth-first-directed(5)
    depth-first-directed(7)
  depth-first-directed(6)
  depth-first-directed(8)
    depth-first-directed(9)

The resulting depth-first spanning forest is shown in Fig. 3.10 together with the *post* values for the reverse topological order. The roots of the trees are always shown at the top.

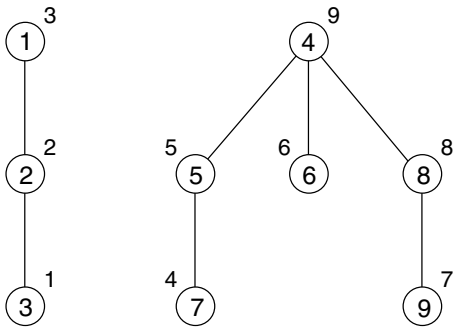


Figure 3.10: On a possible depth-first spanning forest of the graph from Fig. 3.9. The numbers close to the vertices denote the *post* values of the reverse topological order.

If the vertices are treated in a different order, then the resulting depth-first spanning forest might look different. When, e. g., vertex 4 is considered first, then all vertices will be collected in one tree. The resulting tree, assuming that the neighbors of vertex 4 are considered in the order 8,6,5,3, is shown in Fig. 3.11. □

Now we turn back to undirected graphs. A similar algorithm to a depth-first search is an algorithm, which first visits all neighbors of a vertex before proceeding with vertices further away. This algorithm is called a *breadth-first search*. This means that at first all neighbors of the initial vertex are visited. This initial vertex is the root of the *breadth-first spanning tree* being built. These neighbors have distance one from the root, when measured in terms of the number of edges along the shortest path from the root. In the previous example in Fig. 3.8, the edge $\{1, 2\}$ would be included in the spanning tree, if it is constructed using a breadth-first search, as we will see below. In the next step of the algorithm, all neighbors of the vertices treated in the first step are visited, and so on. Thus, a queue[1] can be used to store the vertices

[1] A queue is a linear list, where one adds elements on one side and removes them from the other side.

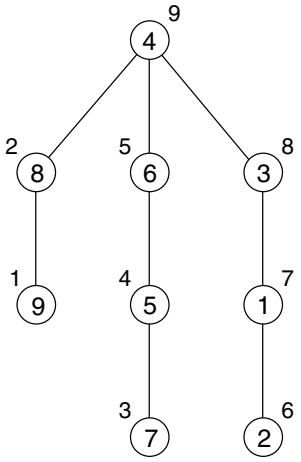


Figure 3.11: Another possible depth-first spanning tree of the graph from Fig. 3.9. The numbers close to the vertices denote the $post$ values of the reverse topological order.

which are to be processed. The neighbors of the current vertex are always stored at the end of the queue. Initially the queue contains only the root. The algorithmic representation reads as follows, $level(i)$ denotes the distance of vertex i from the root r and $pred(i)$ is the predecessor of i in a shortest path from r, which defines the breadth-first spanning tree.

algorithm breadth-first search(G, r, [ref.] $comp, t$)
begin
  Initialize queue $Q := \{r\}$;
  Initialize $level[r] := 0$; $level[i] := -1$ (undefined) for all other vertices;
  $comp[r] := t$;
  Initialize $pred[r] := -1$;
  while Q not empty
  begin
    Remove first vertex i of Q;
    for all neighbors j of i **do**
      if $level[j] = -1$ **then**
      begin
        $level[j] := level[i] + 1$;
        $comp[j] := t$;
        $pred[j] := i$;
        add j at the end of Q;
      end
  end
end

Also for a breadth-first search, for each vertex all neighbors are visited. Thus each edge is again touched twice, which results in a time complexity of $\mathcal{O}(|E|)$. To obtain the breadth-first spanning forest of a graph, one has to call the procedure for all yet unvisited vertices inside a loop over all vertices, as in the algorithm components() presented above.

Example: Breadth-first search

We consider the same graph as in the example above, shown now in Fig. 3.12. Initially the queue contains the source, here 1 again and all values $level[i]$ are "undefined" (-1), except $level[1] = 0$.

$Q = \{0\}, level[1] = 1$

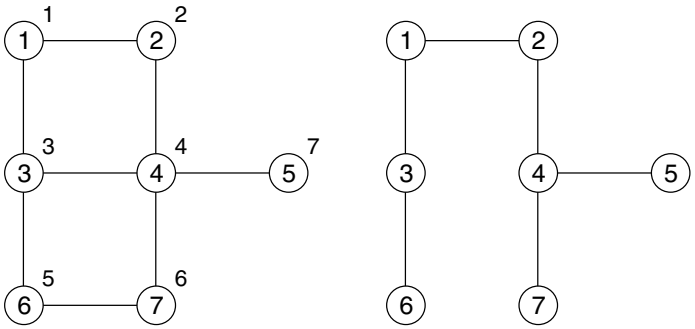


Figure 3.12: A sample graph (left). The number close to the vertices indicate a possible order in which the vertices are visited during a breadth-first search. On the right the resulting breadth-first spanning tree is shown.

While treating vertex 1, its neighbors, vertices 2 and 3, are added to the queue, thus $pred[2] = pred[3] = 1$. They have a distance 1 from the source ($level[2] = level[3] = 1$).

$Q = \{2, 3\}, level[1] = 0, level[2] = level[3] = 1\,.$

Next vertex 2 is processed. It has two neighbors, vertices 1 and 4, but vertex 1 has been visited already ($level[1] \neq -1$), thus only vertex 4 is added to Q with $pred[4] = 2$, $level[4] = level[2] + 1 = 2$. After this iteration the situation is as follows:

$Q = \{3, 4\}, level[1] = 0, level[2] = level[3] = 1, level[4] = 2\,.$

The treatment of vertex 3 adds vertex 6 to the queue ($level[6] = level[3] + 1 = 2$, $pred[6] = 3$). At the beginning of the next iteration vertex 4 is taken. Among its four neighbors, vertices 5 and 7 have not been visited. Thus $level[5] = level[7] = 3$ and $pred[5] = pred[7] = 4$. Now all values of the $pred$ and $level$ arrays are set. Finally, vertices 6,5 and 7 are processed, without any change in the variables.

The resulting breadth-first spanning tree is shown on the right of Fig. 3.12. It is also stored in the array $pred$ of shortest paths to the source, e. g., a shortest

path from vertex 7 to vertex 1 is given by the iterated sequence of predecessors: $pred[7] = 4, pred[4] = 2, pred[2] = 1$. □

3.2.2 Strongly connected component

As we recall from Sec. 3.1.1, the strongly connected components of a directed graph $G = (V, E)$ are the maximal subsets of vertices in which, for each pair i, j, a directed path from i to j and a directed path from j to i, exists in G. As we will show here, they can be obtained by *two* depth-first searches.

The basic idea is as follows. First one performs a depth-first search on G, obtaining the depth-first spanning forest and the reverse topological order in the array $post$. Next, once constructs the *reverse* graph G^R, which is obtained by using the same vertices as in G and with all edges from G reversed

$$G^R \equiv (V, E^R) \quad E^R \equiv \{(j, i) | (i, j) \in E\}\,. \tag{3.1}$$

Then we compute the depth-first spanning forest of G^R, but *we treat the vertices in the main loop in decreasing values of the reverse topological order* obtained in the first computation of the depth-first order of G. In particular, we always start with the vertex having received the highest $post$ number. The trees of the depth-first spanning forest obtained in this way are the strongly connected components of G. The reason is that, for each tree, the vertices with highest $post$ number are those from where all vertices of a depth-first tree can be reached, i. e., the *roots* of the trees. When starting the depth-first search at these vertices for the reverse graph, graph one is able to visit the vertices, from which one is able to reach the root in the original graph. In total these are the vertices from which one can reach the root and which can be reached starting from the root, i. e., they compose the strongly connected components. A detailed proof will be given below. The algorithm can be summarized as follows:

algorithm scc(G)
begin
  initialize *tree*$[i] := 0$ for all $i \in V$;
  $t := 1$; $c := 1$;
  while there is a vertex i with $tree[i]$=0 **do**
    depth-first-directed(G, i, $tree$, t, $post1$, c); t:=t+1
  calculate reverse graph G^R;
  initialize *tree*$[i] := 0$ for all $i \in V$;
  $t := 1$;
  $c := 1$;
  for all $i \in V$ in decreasing value of $post1[i]$ **do**
    if $tree[i]$=0 **do**
      depth-first-directed(G^R, i, $tree$, t, $post2$, c); t:=t+1;
end

Example: Strongly connected component

As an example, we consider the graph from Fig. 3.9. We assume that the reverse topological order is as obtained in the first example run which was shown on page 42, hence the vertices ordered in decreasing $post$ numbers of the reverse topological order are

$$4, 8, 9, 6, 5, 7, 1, 2, 3$$

In this order, the depth-first search is applied to the reverse graph G^R, graph which is shown in Fig. 3.13.

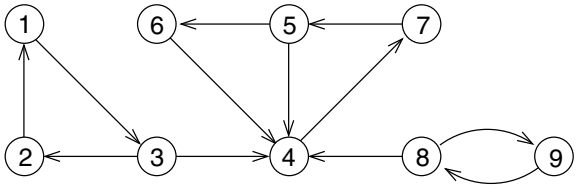
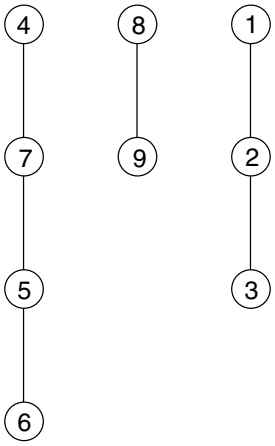


Figure 3.13: The reverse graph of graph shown in Fig. 3.9.

Hence, when calling depth-first-directed(G^R, 4, $tree$, 1, $post2$, c), the vertices 7, 5, and 6 are visited recursively in this order, before the call with $i = 4$ terminates. Next, in the main loop with $t = 2$, the procedure is called for vertex $i = 8$, which has the second highest value in $post1$, the vertices 8 and 9 are processed. For the third ($t = 3$) iteration, depth-first-directed() is called for vertex 1, resulting in the visits of also vertices 3 and 2. The resulting depth-first spanning forest is shown on the right.

This means that we have obtained three trees, corresponding to the three strongly connected components $C_1 = \{1, 2, 3\}$, $C_2 = \{4, 5, 6, 7\}$ and $C_2 = \{8, 9\}$.



□

The algorithm consist of two loops over vertices, containing calls to depth-first-directed(). Each loop takes time $\mathcal{O}(|V| + |E|)$, see Sec. 3.2.1. Note that one can obtain the vertices in decreasing reverse topological order, by writing each vertex in an array after its call to depth-first-directed() has finished. For brevity, we have not included this array in the above algorithm. This means that one can perform the **for** loop in scc() simply by going through the array in reverse order, i. e., without additional time cost. Since also calculating the reverse

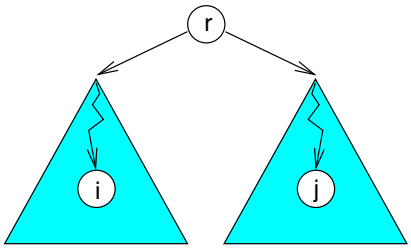
graph graph can be done in $\mathcal{O}(|V| + |E|)$, the full algorithm has a running time $\mathcal{O}(|V| + |E|)$. There are variants of the algorithm, which require only one calculation of the depth-first spanning forest instead of two, also no calculation of the reverse graph occurs there. Instead, they require updating of additional data fields, hence they are a bit harder to understand. The reader interested in these algorithms should consult the standard literature [11–13].

We close this section by showing that the above algorithm does indeed yield the strongly connected components of a directed graph. Note that the strongly connected components of the reverse graph graph are exactly the same as those of the graph itself.

Proof:

($\rightarrow$)
Assume that two vertices i and j are mutually reachable, i. e., are connected by paths. Hence, they are mutually reachable in the reverse graph. This means that during the second calculation of the forest, they will be in the same tree.

($\leftarrow$)
Now we assume that i and j are in the same tree of the depth-first forest calculation of G^R. Let r be the root of this tree, i. e., the vertex for which depth-first-directed() has been called on the top level in the **for** loop. Since the **for** loop is performed in decreasing order of the $post$ values, this means that r has the highest $post$ value of the tree. This means in particular $post[r] > post[i]$ and $post[r] > post[j]$ (*).



We perform the proof by raising a contradiction. We assume that i and j are *not* in the same strongly connected component.

Since i and j are in the same tree with root r for G^R, there must paths from the root r to i and from r to j in the reverse graph graph G^R, hence there must be paths from i to r and from j to r in G. Since i and j are assumed not to be in the same strongly connected component, either there is no path from r to i in G or there is no path from r to j in G.

Without loss of generality, we consider the first case $r \not\rightarrow i$. There are two possibilities:

a) i is visited before r in the calculation of the depth-first forest of G. But then, because there is a path from i to r in G, the call to depth-first-directed() for r would finish before the call for i, hence we would have $post[i] > post[r]$. This is a contradiction to (*)!

b) r is visited before i. Since there is no path from r to i, vertex i will still be unvisited, when the call for r has been finished, hence we will have again $post[i] > post[r]$. This is a contradiction to (*)!

In the same way, we can raise a contradiction when assuming that path $r \rightarrow j$ does not exist in G. This means in total that i and j must be in the same strongly-connected component. QED

3.2.3 Minimum spanning tree

The algorithm used to construct a minimum spanning tree of a given graph (see Sec. 3.1.3 for a definition) is *greedy*. This means that at every step the algorithm takes the least expensive decision. In contrast to the TSP-algorithm presented in Chap. 2, a global minimum is guaranteed to be found. The basic idea of Prim's algorithm is that one starts with the edge of minimal weight, and goes on adding minimal edges connected to the already constructed subtree, unless a cycle would result:

algorithm Prim(G)
begin
  choose $\{i,j\} : \omega_{ij} := \min\{\omega_{km} | \{k,m\} \in E\}$;
  $S := \{i,j\}$;
  $\overline{S} := V \setminus \{i,j\}$;
  $T := \{\{s,r\}\}$;
  $X := \omega_{ij}$;
  while $|S| < |V|$ **do**
  begin
    choose $\{i,j\} : i \in S,\ j \in \overline{S}$ and $\omega_{ij} := \min\{\omega_{km} | k \in S, m \in \overline{S}, \{k,m\} \in E\}$;
    $S := S \cup \{j\}$;
    $\overline{S} := \overline{S} \backslash \{j\}$;
    $T := T \cup \{\{i,j\}\}$;
    $X := X + \omega_{ij}$;
  end
  return (S,T);
end

The action of Prim's algorithm is illustrated in Fig. 3.14. One can easily verify that it produces in fact a minimum spanning tree: Imagine, that this is not the case, i. e., in the algorithm you have, for the first time, added a wrong edge $\tilde{e}$ at the kth step. Now imagine that you have a minimum spanning tree coinciding with the tree constructed in Prim's algorithm in the first $k-1$ selected edges, but not containing $\tilde{e}$. If you add edge $\tilde{e}$ to the minimum spanning tree, you introduce a cycle, which, by Prim's algorithm, contains at least one edge of higher weight. Deleting the latter thus gives a new spanning tree of smaller weight, which is a contradiction to the minimality of the initial spanning tree.

Since each edge is chosen at most one, the while loop is performed $\mathcal{O}(|E|)$ times. In a simple version, choosing the edge with minimum weight requires a loop over all edges ($\mathcal{O}(|E|)$), while a more refined version could use a *priority queue* [11–13], such that the choose operation takes only $\mathcal{O}(\log|E|)$ time, leading to a total of $\mathcal{O}(|E|\log|E|)$ running time.

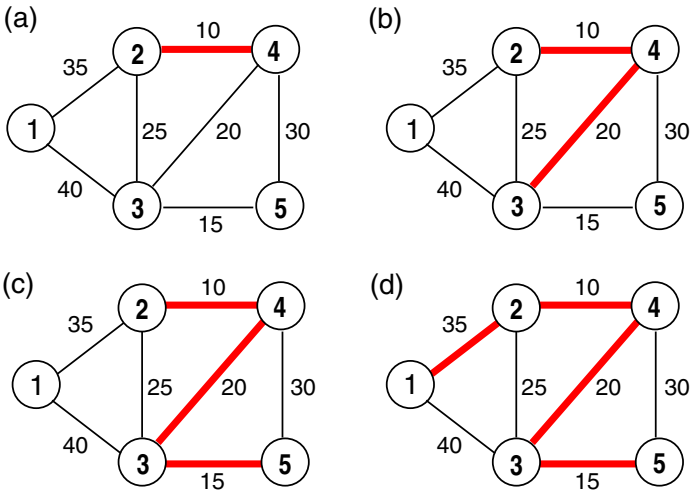


Figure 3.14: Illustration of Prim's algorithm. The graph contains five vertices numbered from 1 to 5. The numbers along the edges denote the edge weights. The current edges contained in the spanning tree are shown in bold. From (a), starting with the lowest cost edge, to (d), successively new edges are added until a minimum spanning tree is reached.

3.3 Random graphs

3.3.1 Two ensembles

The basic idea of random-graph theory is to study *ensembles* of graphs instead of single, specific graph instances. Using probabilistic tools, it is sometimes easier to show the existence of graphs with specific properties, or, vice versa, to look at how a *typical* graph with given properties (e. g., order, size, degrees…) appears.

The simplest idea goes back to a seminal paper published by Erdős and Rényi in 1960 [14]. They assumed all graphs of the same order N and size M to be equiprobable. There are mainly two slightly different ensembles used in the field:

The ensemble $\mathcal{G}(N, M)$ contains all graphs of N vertices and M edges. The measure is flat, i. e., every graph has the same probability, see, e. g., Fig. 3.15. Note that all graphs obtained from a given graph by permuting the vertices are different graphs, i. e., they are counted individually.

The ensemble $\mathcal{G}(N, p)$, with $0 \leq p \leq 1$, contains graphs with N vertices. For every vertex pair i, j an edge $\{i, j\}$ is drawn independently with probability p. For $p = 0$, the graph has no

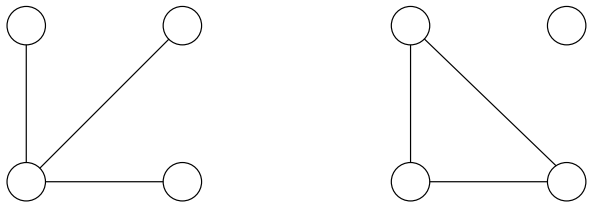


Figure 3.15: These two graphs are equiprobable in $\mathcal{G}(4,3)$, even if their structure is quite different. Whereas the left graph is a connected tree, the right one is not connected and contains a cycle.

edges, for $p = 1$, the graph is complete (K_N). On average, the number of edges for given p is

$$\overline{M} = p\binom{N}{2} = p\frac{N!}{(N-2)!2!} = p\frac{N(N-1)}{2} \tag{3.2}$$

where the over-bar denotes the average over $\mathcal{G}(N,p)$. This ensemble is analytically easier to handle, so we mainly work with $\mathcal{G}(N,p)$.

The specific case $\mathcal{G}(N,1/2)$ can also be considered as *the* random ensemble of graphs: All graphs of N vertices are equiprobable, independently of their edge numbers.

One important type of statement is that a $\mathcal{G}(N,p)$-graph fulfils *almost surely* (or *with probability one*) some condition C. This expression means that, for $N \to \infty$, the probability that a graph drawn from $\mathcal{G}(N,p)$ fulfils this condition C, converges to one.

3.3.2 Evolution of graphs

Sometimes, it is very instructive to imagine the ensemble $\mathcal{G}(N,p)$ via an evolutionary process of graphs of N vertices which, with time, get more and more edges. This can be realized in the following way. We take $V = \{1,\ldots,N\}$, and for all $i,j \in V$, $i < j$, we independently draw a random number x_{ij} being equally distributed in $(0,1)$. Initially the graph has no edges. Now, we start to grow $p(t)$. Whenever it exceeds a number x_{ij}, an edge between vertices i and j is added. Thus, at time t, the graph belongs to $\mathcal{G}(N,p(t))$. There are some interesting stages in this evolution:

- $p(t) \sim 1/N^2$: The first isolated edges appear.
- $p(t) \sim 1/N^{3/2}$: The first vertices have degree 2, i. e., the first edges have a common vertex.
- $p(t) \sim 1/N^{\alpha}$, any $\alpha > 1$: The graph is almost surely a forest.
- $p(t) \sim 1/N$: The average vertex degree stays finite for $N \to \infty$, first cycles appear, the first macroscopic (i. e., of order $\mathcal{O}(N)$) subgraph appears, macroscopic q-cores appear.

- $p(t) \simeq \ln(N)/N$: The graph becomes connected.
- $p(t) \simeq (\ln(N) + \ln(\ln(N))/N$: The graph becomes Hamiltonian.

For the proof see the book by Bollobás mentioned on page 25.

3.3.3 Finite-connectivity graphs: The case $p = c/N$

The most interesting case is given by $p = c/N$, where the medium size of the graph $\overline{M} = c(N-1)/2$ grows linearly with the graph order N. In this section, we first discuss the fluctuations of the total edge number M, and some local properties like degrees and the existence of small cycles. In the following two sections we finally switch to global properties. We discuss *random-graph percolation*which describes a phase transition from graphs having only small connected components, even for $N \to \infty$, to graphs also having one giant component which unifies a finite fraction of all vertices, i. e., whose order also grows linearly with the graph order N. The last subsection discusses the sudden emergence of a q-core.

The number of edges

For a graph from $\mathcal{G}(N, c/N)$, every pair of vertices becomes connected by an edge with probability c/N, and remains unlinked with probability $1 - c/N$. In contrast to the $\mathcal{G}(N, M)$-ensemble, the total number of edges is a random variable and fluctuates from sample to sample. Let us therefore calculate the probability P_M of it having exactly M edges. This is given by

$$P_M = \binom{N(N-1)/2}{M} \left[\frac{c}{N}\right]^M \left[1 - \frac{c}{N}\right]^{N(N-1)/2-M} . \tag{3.3}$$

The combinatorial prefactor describes the number of possible selections of the M edges out of the $N(N-1)/2$ distinct vertex pairs, the second factor gives the probability that they are in fact connected by edges, whereas the last factor guarantees that there are no further edges. In the limit of large N, and for $M \sim N$, where $M \ll N(N-1)/2$, we use

$$\left(\frac{N(N-1)}{2}\right)\left(\frac{N(N-1)}{2} - 1\right)\cdots\left(\frac{N(N-1)}{2} - M + 1\right) = \left(\frac{N(N-1)}{2}\right)^M + \mathcal{O}\left(N^{2(M-1)}\right) \tag{3.4}$$

and $(1 - c/N)^{zN} \approx \exp(-cZ)$, hence the above expression can be asymptotically approximated by

$$P_M \simeq \frac{(N(N-1)/2)^M}{M!} \left[\frac{c}{N}\right]^M \exp(-c(N-1)/2) . \tag{3.5}$$

Plugging in $\overline{M} = c(N-1)/2$, this asymptotically leads to a *Poissonian distribution* with mean $\overline{M}$,

$$P_M \simeq \exp\left(-\overline{M}\right) \frac{\overline{M}^M}{M!} \,. \tag{3.6}$$

The fluctuations of the edge number M can be estimated by the *standard deviation* of P_M,

$$\sigma(M) = \sqrt{\overline{(M-\overline{M})^2}} = \sqrt{\overline{M^2} - \overline{M}^2} \,. \tag{3.7}$$

We first calculate

$$\begin{aligned}
\overline{M^2} &= \overline{M} + \overline{M(M-1)} \\
&= \overline{M} + \sum_{M=0}^{\infty} M(M-1) P_M \\
&= \overline{M} + \exp\left(-\overline{M}\right) \sum_{M=2}^{\infty} \frac{\overline{M}^M}{(M-2)!} \\
&= \overline{M} + \overline{M}^2 \exp\left(-\overline{M}\right) \sum_{m=0}^{\infty} \frac{\overline{M}^m}{m!} \\
&= \overline{M} + \overline{M}^2 \,.
\end{aligned} \tag{3.8}$$

In the third line, we have eliminated the cases $M = 0, 1$ which vanish due to the factor $M(M-1)$, in the fourth line we have introduced $m = M + 2$ which runs from 0 to ∞. The sum can be performed and gives $\exp(\overline{M})$. Using Eq. (3.7) we thus find

$$\sigma(M) = \sqrt{\overline{M}} \,, \tag{3.9}$$

which is a special case of the *central limit theorem*. The relative fluctuations $\sigma(M)/\overline{M} = 1/\sqrt{\overline{M}}$ decay to zero for $\overline{M} = c(N-1)/2 \to \infty$, see Fig. 3.16. In this limit, the sample-to-sample fluctuations of the edge number become less and less important, and the ensemble $\mathcal{G}(N, c/N)$ can be identified with $\mathcal{G}(N, \overline{M})$ for most practical considerations.

The degree distribution

Let us now discuss the degrees of the graph. We are interested in the probability p_d that a randomly chosen vertex has exactly degree d. The set of all p_d is called the *degree distribution*. It can be easily calculated:

$$p_d = \binom{N-1}{d} \left[\frac{c}{N}\right]^d \left[1 - \frac{c}{N}\right]^{N-d-1} \,. \tag{3.10}$$

The meaning of the terms on the right-hand side is the following: The factor $\binom{N-1}{d}$ enumerates all possible selections for d potential neighbors. Each of these vertices is connected to

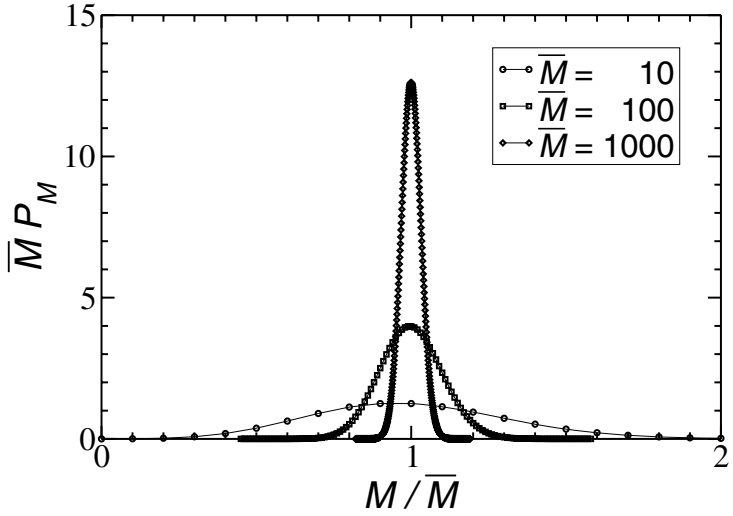


Figure 3.16: The distribution of edge numbers for $\overline{M} = 10, 100, 1000$, rescaled by $\overline{M}$. The distributions obviously sharpen for increasing $\overline{M}$.

the central vertex with probability c/N, whereas the other $N-d-1$ vertices are not allowed to be adjacent to the central one, i. e., they contribute a factor $(1-c/N)$ each. In the large-N limit, where any fixed degree d is small compared with the graph order N, we can continue in an analogous way to the last subsection and find

$$\begin{aligned} p_d &= \lim_{N\to\infty} \frac{(N-1)(N-2)\cdots(N-d)}{N^d} \Big[1-\frac{c}{N}\Big]^{N-d-1} \frac{c^d}{d!} \\ &= e^{-c}\frac{c^d}{d!}\,, \end{aligned} \tag{3.11}$$

i. e., also the degrees are distributed according to a Poissonian distribution. It is obviously normalized, and the average degree is

$$\begin{aligned} \sum_{d=0}^{\infty} d\,p_d &= \sum_{d=1}^{\infty} e^{-c}\frac{c^d}{(d-1)!} \\ &= c\,. \end{aligned} \tag{3.12}$$

This was clear since the expected number of neighbors to any vertex is $p(N-1) \to c$. Note also that the fluctuation of this value are again given by the standard deviation $\sigma(c) = \sqrt{c}$, which, however, remains comparable to c if $c = \mathcal{O}(1)$. The degree fluctuations between vertices thus also survive in the thermodynamic limit, there is, e. g., a fraction e^{-c} of all vertices which is completely isolated.

If we randomly select an edge and ask for the degree of one of its end-vertices, we obviously find a different probability distribution q_d, e. g., degree zero cannot be reached ($q_0 = 0$). This probability is obviously proportional to p_d as well as to d, by normalization we thus find

$$q_d = \frac{dp_d}{\sum_d dp_d} = \frac{dp_d}{c} = e^{-c}\frac{c^{d-1}}{(d-1)!} \tag{3.13}$$

for all $d > 0$. The average degree of vertices selected in this way equals $c + 1$, it includes the selected edge together with, on average, c additional *excess edges*. The number $d - 1$ of these additional edges will be denoted as the *excess degree* of the vertex under consideration.

Cycles and the locally tree-like structure

The degrees are the simplest local property. We may go slightly beyond this and ask for the average number of small subgraphs. Let us start with subtrees of k vertices which thus have $k - 1$ edges. We concentrate on labeled subtrees (i. e., the order in which the vertices appear is important) which are not necessarily induced (i. e., there may be additional edges joining vertices of the tree which are not counted). Their expected number is proportional to

$$N(N-1)\cdots(N-k+1)\left[\frac{c}{N}\right]^{k-1} \quad = \quad Nc^{k-1} + \mathcal{O}(1)\,, \tag{3.14}$$

combinatorial factors specifying $k - 1$ specific edges out of the $k(k-1)/2$ possible ones, are omitted. This number thus grows linearly with the graph order N. If, on the other hand, we look to cycles of finite length k, we have to find k edges. The above expression takes an additional factor c/N, the expected number of cycles is thus of $\mathcal{O}(1)$, i. e., the number of triangles, squares, etc. stays finite even if the graph becomes infinitely large! This becomes more drastic if one looks to cliques K_k with $k \geq 4$, these become more and more likely to be completely absent if N becomes large. Thus, if we look locally to induced subgraphs of any finite size k, these are almost surely trees or forests. This property is denoted as *locally tree-like*.

There are, however, loops, but these are of length $\mathcal{O}(\ln N)$ [15]. In the statistical physics approach, we will see that these loops are of fundamental importance, even if they are of diverging length.

Another side remark concerns the dimensionality of the graph, i. e., the possibility of "drawing" large graphs in a finite-dimensional space. If you look to any D-dimensional lattice, the number of neighbors up to distance k grows as k^D. On the other hand, in a tree of fixed average degree, this number is growing exponentially! In this sense, random graphs have to be considered to be infinite dimensional. This sounds strange at first, but finally explains the analytical tractability which will be observed later in this book.

3.3.4 The phase transition: Emergence of a giant component

From Sec. 3.3.2 we know that random graphs with $p = c/N$ are almost surely not connected. What can we say about the components?

Having in mind the growth process described above, we may imagine that for small c there are many small components, almost all of them being trees. If c increases, we add new edges and some of the previously disconnected components now become connected. The number of components decreases, the number of vertices (order) of the components grows. Concentrating on the largest component $L^{(0)}(G)$ of a graph G, the following theorem was demonstrated in 1960 by Erdős and Rényi [14]:

Theorem: *Let $c > 0$ and G_c drawn from $\mathcal{G}(N, c/N)$. Set $\alpha = c - 1 - \ln c$.*
(i) If $c < 1$ then we find almost surely

$$|L^{(0)}(G_c)| = \frac{1}{\alpha} \ln N + \mathcal{O}(\ln \ln N)$$

(ii) If $c > 1$ we almost surely have

$$|L^{(0)}(G_c)| = \gamma N + \mathcal{O}(N^{1/2})$$

with $0 < \gamma = \gamma(c) < 1$ being the unique solution of

$$1 - \gamma = e^{-c\gamma} .$$

All smaller components are of order $\mathcal{O}(\ln N)$.

This theorem makes a very powerful statement: As long as $c < 1$, all components have order up to $\mathcal{O}(\ln N)$. This changes markedly if $c > 1$. There appears one *giant component* connecting a finite fraction of all vertices, but all the other components are still small compared to the giant one, they have only $\mathcal{O}(\ln N)$ vertices. This means that at $c = 1$ the system undergoes a *phase transition*. In contrast to physical phase transitions, it is not induced by a change in temperature, pressure or other *external* control parameters, but by the change of the average vertex degree c, i. e., by a *structural* parameter of the graph. Due to the obvious analogy to percolation theory [16], this phase transition is also called *random-graph percolation* in the literature.

This theorem is one of the most fundamental results of random-graph theory, but we do not present a complete proof. There is, however, a simple argument that the component structure changes at $c = 1$. It is based on the locally tree-like structure of all components, and is presented in the limit $N \to \infty$.

If we select any vertex, on average it will have c neighbors. According to Eq. (3.13), each of these will have c additional neighbors, the first vertex thus has, on average, c^2 second neighbors. Repeating this argument, we conclude that the expected number of kth neighbors equals c^k (A vertex j is called a kth neighbor of a vertex i if the minimum path in the graph G connecting both contains exactly k edges).

Now we can see the origin of the difference for $c < 1$ and $c > 1$. In the first case, the prescribed process decreases exponentially, and it is likely to die out after a few steps. If, in contrast, we have $c > 1$, the expected number of kth neighbors grows exponentially. Still, there is a finite probability that the process dies out after a few steps (e. g., e^{-c} if dying in the first step, i. e., if there are no neighbors at all), but there is also a finite probability of proceeding for ever.

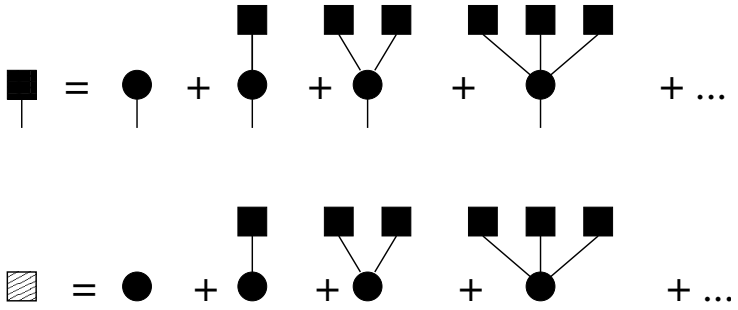


Figure 3.17: Schematic representation of the iterative solution for the probability that a vertex does not belong to the giant component. The first line shows the self-consistent equation for a vertex reached by a random edge: The black square with the half-edge represents the probability that the vertex reached is not connected to the giant component by one of its excess edges. This can happen because it has no further neighbors, or it is connected to one, two, etc., vertices not connected with the giant component. The second line shows the resulting equation for a randomly selected vertex, as represented by the shaded square.

This argument can be made more rigorous by considering the following iterative construction: Fist we calculate the probability π, that a randomly selected end-vertex of a randomly selected link is not connected via other edges with the giant component of the graph. In Figure 3.17 this is represented by a black square connected to a half-edge. This vertex can either have degree one, i. e., it has no further incident edges which would be able to connect it to the giant component, or it is connected to other vertices which themselves are not connected to the giant component by their excess edges. Having in mind that almost all small connected components are trees, these neighbors are connected with each other only via the selected vertex, and the probability that d neighbors are not connected to the giant component equals simply π^d. Using the probability distribution q_d Eq. (3.13) of degrees of vertices reached by random edges, we thus find:

$$\begin{aligned} \pi &= q_1 + q_2\pi + q_3\pi^2 + q_4\pi^3 + \dots \\ &= \sum_{d=1}^{\infty} e^{-c}\frac{c^{d-1}}{(d-1)!}\,\pi^{d-1} = e^{-c}\sum_{d'=0}^{\infty}\frac{(c\pi)^{d'}}{(d')!} \\ &= e^{-c(1-\pi)}\,. \end{aligned} \tag{3.15}$$

This quantity can thus be determined self-consistently for every value of c, and allows us to calculate the probability $1-\gamma$ that a randomly selected vertex does not belong to the giant component. Applying similar arguments to before, this vertex can either be isolated, or

connected only to other vertices which, via their excess edges, are not connected to the giant component, see the second line of Fig. 3.17. We thus find

$$\begin{aligned} 1-\gamma &= p_0 + p_1\pi + p_2\pi^2 + p_3\pi^3 + \dots \\ &= \sum_{d=1}^{\infty} e^{-c}\frac{c^d}{d!}\,\pi^d \\ &= e^{-c(1-\pi)}\,. \end{aligned} \tag{3.16}$$

From these equations we see first that, for our random graph, $\pi = 1-\gamma$. Plugging this into the last line, we obtain the equation

$$1-\gamma = e^{-c\gamma} \tag{3.17}$$

as given in the theorem.

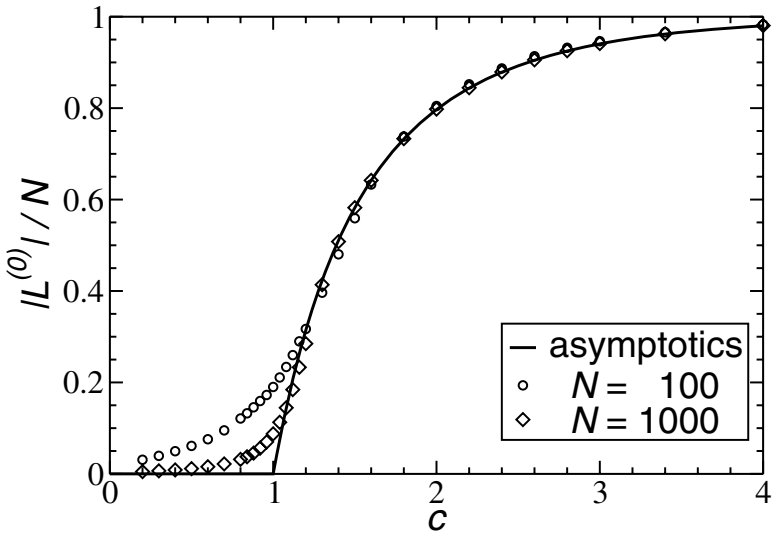


Figure 3.18: Fraction of vertices belonging to the largest component of a random graph, as a function of the average degree c. The symbols are numerical data for $N = 100, 1000$, averaged always over 100 randomly generated graphs. The full line gives the asymptotic analytical result: Below $c = 1$, the largest component is sub-extensive, above it is extensive.

Let us shortly discuss the shape of $\gamma(c)$. For $c = 1+\varepsilon$, with $0 < \varepsilon \ll 1$, we also expect γ to be small, and expand the above equation to second order in γ:

$$1-\gamma = 1-(1+\varepsilon)\gamma + \frac{1}{2}(1+\varepsilon)^2\gamma^2 + \mathcal{O}(\gamma^3)\,. \tag{3.18}$$

The term $1-\gamma$ cancels on both sides. Dividing by γ, and keeping only the first order in ε and γ, we thus find

$$\gamma = 2\varepsilon \; . \tag{3.19}$$

The relative order of the giant component thus starts to grow linearly in $c-1$ (we say the critical exponent for the giant component equals one), and converges exponentially fast to 1 for larger c. The full curve is given in Fig. 3.18, together with numerical data for finite random graphs.

3.3.5 The emergence of a giant q-core

In Sec. 3.1.5, we have introduced the q-core of a graph as the maximal subgraph having minimum vertex degree of at least q. Quite recently, the problem of the existence of a q-core in random graphs was solved in a beautiful paper by B. Pittel, J. Spencer and N. C. Wormald [17] by analyzing exactly the linear-time algorithm given there. Here we will give an easily accessible summary of their approach, not including the technical details of the mathematical proofs.

Let us first start with a numerical experiment: We have generated medium-size random graphs with $N = 50\,000$ vertices and various average degrees c, and we have applied the q-core algorithm for $q = 2$ and $q = 3$. The results for $q = 2$ are not very astonishing: Up to $c = 1$, the 2-core is very small, whereas it starts to occupy a finite fraction of all vertices for $c > 1$. This is consistent with the results on the giant component obtained in the last section: As long as the latter does not exist, almost all vertices are collected in finite trees, and thus do not belong to the 2-core. A small difference in the emergence of the giant component can, however, be seen looking in the vicinity of $c = 1$. Whereas the giant component starts to grow linearly with the distance from the critical point, see Eq. (3.19), the 2-core emerges with zero slope, see Fig. 3.19. This means that the critical exponents of both processes appear to differ from each other.

The situation changes dramatically for $q \geq 3$. At the corresponding threshold $c(q)$, which is monotonously increasing with q, the q-core appears *discontinuously* – immediately unifying a finite fraction of all vertices. For $q = 3$, the threshold is found to be $c(3) \simeq 3.35$. Slightly below this average degree, almost all graphs have no extensive q-core at all. At the transition, the order of the 3-core jumps abruptly to about $0.27N$, i. e., it contains about 27% of all vertices!

Wormald's method

The method of describing this behavior was presented mathematically by Wormald [18], it is, however, well-known and applied in physics under the name of *rate equations*. It analyzes a slightly modified version of the q-core algorithm. In every algorithmic step, only one vertex of degree smaller than q is selected randomly and removed from the graph. The quantity

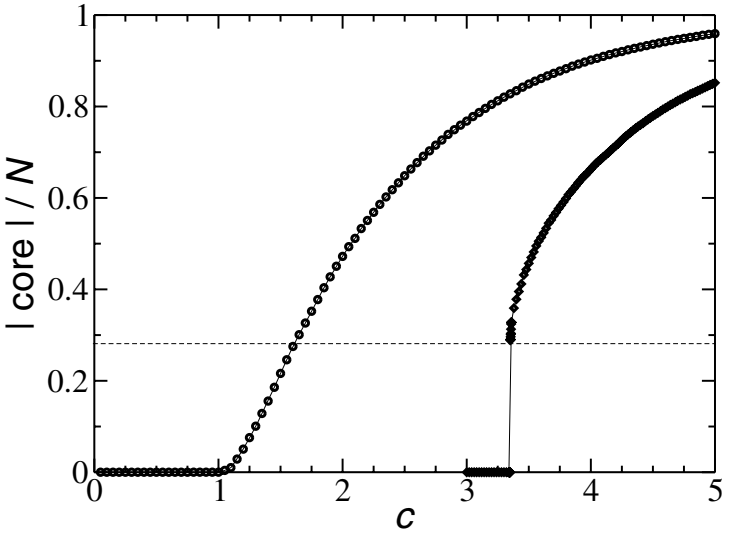


Figure 3.19: Fraction of vertices belonging to the q-core for $q = 2$ (circles) and $q = 3$ (diamonds). The numerical data result from a single graph of $N = 50\,000$ vertices. The vertical line gives the theoretical value of the 3-core size at the threshold $c = 3.35$.

calculated in the analysis of Pittel *et al.* is the degree distribution p_d, as it changes under the algorithmic decimation. This is obviously a perfect candidate for determining the halting point of the algorithm. If $p_d = 0$ for all $d < q$, the remaining subgraph is the q-core.

An important point in the analysis, which will not be justified mathematically here, is that the graph dynamics is *self-averaging* in the thermodynamic limit $N \to \infty$. After a certain number $T = tN = \mathcal{O}(N)$ of algorithmic steps almost all random graphs have the same degree distribution, which is also almost surely independent of the random order of decimated vertices. Technically, this allows us to average over both kinds of randomness and to determine the average, or expected action of the algorithm.

In the last paragraph, we have already introduced a rescaling of the number of algorithmic steps by introducing the *time* $t = T/N$. This quantity remains finite even in the thermodynamic limit, running initially from $t = 0$ to at most $t = 1$, where the full graph would have been removed. Further on, this quantity advances in steps $\Delta t = 1/N$, and thus becomes continuous in the large-N limit. The last observation will allow us to work with differential equations instead of discrete-time differences.

After these introductory remarks we can start the analysis by first noting that the total vertex number evolves as $N(T) = (1-t) \cdot N$, since we remove one vertex per step. Introducing later on the numbers $N_d(T)$ of vertices having d neighbors at time $t = T/N$, and the corresponding

degree distribution $p_d(t) = N_d(T)/N(T)$, we can write down the *expected* change of N_d in the $(T+1)$st step. It contains two different contributions:

- The first contribution concerns the removed vertex itself. It appears only in the equations for N_d with $d < q$, because no vertex of higher current degree is ever removed.
- The second contribution results from the neighbors of the removed vertex. Their degree decreases by one.

We thus find the equation below where we explain the meaning of the terms in detail:

$$N_d(T+1) - N_d(T) = -\frac{\chi_d p_d(t)}{\overline{\chi}} + \frac{\overline{d\chi}}{\overline{\chi}} \left[-\frac{d\, p_d(t)}{c(t)} + \frac{(d+1)p_{d+1}(t)}{c(t)} \right] \tag{3.20}$$

which are valid for all degrees d. Here we have introduced χ_d as an indicator for vertices of degree at most $q-1$, i. e., it equals one if $d < q$ and zero else. The overbar denotes as before the average over the graph, i. e., $\overline{\chi} = \sum_d \chi_d p_d(t)$ and $\overline{d\chi} = \sum_d d\chi_d p_d(t)$. Note that these averages are now, even if not explicitly stated, time dependent. We keep the explicit notation for $c(t) \equiv \overline{d} = \sum_d d p_d(t)$. All terms have the following form: they are products of how often the corresponding process happens on average in one step, and of the probability that the corresponding process affects a vertex of degree d. The first term of the right-hand side of Eq. (3.20) describes the removal of the selected vertex (i. e., this process happens exactly once within each step), the indicator χ_d guarantees that only vertices of degree $d < q$ can be removed. The second term contains the neighbors: On average, there are $\overline{d\chi}/\overline{\chi}$ neighbors, each of them having degree d with probability $q_d(t) = d p_d(t)/c(t)$. A loss term stems from vertices having degree d before removing one of their incident edges, a gain term from those having degree $d+1$.

In the thermodynamic limit, the difference on the left-hand side of Eq. (3.20) becomes a derivative:

$$\begin{aligned} N_d(T+1) - N_d(T) &= \frac{(1-t+\Delta t)p_d(t+\Delta t) - (1-t)p_d(t)}{\Delta t} \\ &= \frac{d}{dt}\left\{(1-t)p_d(t)\right\} \ . \end{aligned} \tag{3.21}$$

The last two equations result in a closed set of equations for the degree distribution,

$$\frac{d}{dt}\left\{(1-t)p_d(t)\right\} = -\frac{\chi_d p_d(t)}{\overline{\chi}} + \frac{\overline{d\chi}}{\overline{\chi}} \left[-\frac{d\, p_d(t)}{c(t)} + \frac{(d+1)p_{d+1}(t)}{c(t)} \ , \right] \tag{3.22}$$

which will be solved in the following. First we derive some global equations for averages over $p_d(t)$. The simplest one can be obtained by summing Eq. (3.22) over all d, which gives the trivial consistency relation $\frac{d}{dt}(1-t) = -1$. A much more interesting equation results by first multiplying Eq. (3.22) by d, and then summing over all d. Introducing the short-hand notation

$m(t) = (1-t)c(t)$, we find

$$\begin{aligned}\dot{m}(t) &= \frac{d}{dt}\left\{(1-t)\sum_d dp_d(t)\right\} \\ &= \frac{\overline{d\chi}}{\overline{\chi}}\left[-1-\frac{\overline{d^2}}{c(t)}+\frac{\overline{d(d-1)}}{c(t)}\right] \\ &= -2\frac{\overline{d\chi}}{\overline{\chi}}\,. \end{aligned} \tag{3.23}$$

The initial condition is hereby $m(t=0) = c$.

The main problem in solving Eqs (3.21) is, however, that they form an infinite set of nonlinear differential equations. A direct solution is therefore far from obvious. Our salvation lies in the fact that the algorithm never directly touches a vertex of degree $d \geq q$. These vertices change their properties only via the random removal of one of their incident edges. Therefore they maintain their random connection structure, and their Poissonian shape, only the effective connectivity changes. This can be verified using the ansatz, with $\beta(t)$ being the effective connectivity ($\beta(0) = c$):

$$(1-t)p_d(t) = \frac{N_d(T)}{N} \stackrel{!}{=} e^{-\beta(t)}\frac{\beta(t)^d}{d!}\,, \qquad \forall\, d \geq q\,. \tag{3.24}$$

This ansatz introduces a radical simplification. An infinite number of probabilities is parametrized by one single parameter $\beta(t)$. The validity of this ansatz can be justified by plugging it into Eq. (3.21) for arbitrary d. We obtain for the left-hand side

$$\text{l.h.s.} = \dot{\beta}(t)\left[e^{-\beta(t)}\frac{\beta(t)^{d-1}}{(d-1)!} - e^{-\beta(t)}\frac{\beta(t)^d}{d!}\right] \tag{3.25}$$

and for the right-hand side

$$\text{r.h.s.} = \frac{\overline{d\chi}}{\overline{\chi}}\left[-e^{-\beta(t)}\frac{\beta(t)^d}{(d-1)!(1-t)c(t)} + e^{-\beta(t)}\frac{\beta(t)^{d+1}}{d!(1-t)c(t)}\right]\,. \tag{3.26}$$

They have to be equal for all $d \geq q$, and we can compare the components of the monomial of the same order of both sides:

$$\dot{\beta}(t) = -\frac{\beta(t)}{m(t)}\frac{\overline{d\chi}}{\overline{\chi}}\,. \tag{3.27}$$

The equations thus become closed under ansatz (3.24), and instead of having an infinite number of equations for the $p_d(t)$, $d = q, q+1, \ldots$, just one equation for $\beta(t)$ remains. Interestingly, the q-dependence in this equation is dropped.

If we compare Eqs (3.23) and (3.27), we find

$$2\frac{\dot{\beta}(t)}{\beta(t)} = \frac{\dot{m}(t)}{m(t)} \tag{3.28}$$

which, using the initial conditions, is solved by

$$m(t) = \frac{\beta(t)^2}{c} \,. \tag{3.29}$$

The function $m(t)$ can thus be replaced in Eq. (3.27), and we get

$$\dot{\beta}(t) = -\frac{c}{\beta(t)} \frac{\overline{d\chi}}{\overline{\chi}} \,. \tag{3.30}$$

As a further step, we still have to remove $\overline{\chi}$ and $\overline{d\chi}$ from the last equation. This can be obtained by using normalization, when applying Eq. (3.24)

$$\begin{aligned} 1 &= \sum_{d=0}^{\infty} p_d(t) \\ &= \sum_{d=0}^{\infty} \chi_d p_d(t) + \sum_{d=q}^{\infty} \frac{1}{1-t} e^{-\beta(t)} \frac{\beta(t)^d}{d!} \\ &= \overline{\chi} + \frac{1}{1-t} F_q(\beta(t)) \,. \end{aligned} \tag{3.31}$$

In the same way, we obtain for the average degree $c(t)$:

$$F_q(\beta) := 1 - \sum_{d=0}^{q-1} e^{-\beta} \frac{\beta^d}{d!} \,, \tag{3.32}$$

and

$$\begin{aligned} c(t) &= \sum_{d=0}^{\infty} d p_d(t) \\ &= \sum_{d=0}^{\infty} d\chi_d p_d(t) + \sum_{d=q}^{\infty} \frac{d}{1-t} e^{-\beta(t)} \frac{\beta(t)^d}{d!} \\ &= \overline{d\chi} + \frac{\beta(t)}{1-t} F_{q-1}(\beta(t)) \,. \end{aligned} \tag{3.33}$$

Inserting this into Eq. (3.30), and using Eq. (3.29) to eliminate $c(t) = m(t)/(1-t)$, the equation for $\beta(t)$ finally becomes:

$$\dot{\beta}(t) = -\frac{\beta(t) - cF_{q-1}(\beta(t))}{1 - t - F_q(\beta(t))} \,. \tag{3.34}$$

We have thus succeeded in reducing the infinite set (3.22) into the single Eq. (3.34)! Still, this equation is non-linear and thus not easily solvable for arbitrary q. The important information about the q-core can, however, be obtained even without solving this equation for all times, but by determining only the halting point of the algorithm – which is the q-core of the input graph.

Assume that the graph has a giant q-core. It contains $N(t_f) = (1 - t_f)N$ vertices, where $0 < t_f < 1$ is the halting time of the algorithm. At this point, all remaining vertices have degree $d \geq q$, we thus have $\overline{\chi} = \overline{d\chi} = 0$ and $\beta_f = \beta(t_f) > 0$. According to (3.31,3.33) we therefore have

$$\begin{aligned} 1 - t_f &= F_q(\beta_f) \\ \frac{\beta_f}{c} &= F_{q-1}(\beta_f) \, . \end{aligned} \tag{3.35}$$

The second equation fixes β_f, whereas t_f itself and thus the order of the giant q-core follow directly from the first equation. If there are two or more solutions for β_f one has to choose the largest one which is smaller than c. The initial condition for $\beta(t)$ is $\beta(0) = c$, and Eq. (3.34) results in a negative slope, i. e., in a decrease of $\beta(t)$ with time.

The case $q = 2$

Let us first consider the case of $q = 2$ colors only. From numerical experiments we have seen that the 2-core of a random graph seems to set in continuously at average degree $c = 1$. Can we confirm this, starting with Eqs (3.35)?

The self-consistent equation for $\beta_f = \beta(t_f)$ becomes

$$\frac{\beta_f}{c} = 1 - e^{-\beta_f} \, , \tag{3.36}$$

as represented graphically in Fig. 3.20. The right-hand side starts in $\beta = 0$ with slope one, and asymptotically tends to one. The left-hand side is a straight line of slope $1/c$. For $c < 1$ the only solution of Eq. (3.36) is thus given by $\beta_f = 0$, which according to the first of Eqs (3.35) corresponds to $1 - t_f = 0$, and the 2-core has vanishing size. At $c > 1$, the right-hand side has a smaller slope in the origin, but still diverges for large β. The two curves therefore have a second crossing point $\beta_f > 0$ which, according to the discussion at the end of the last section, has to be selected as the relevant solution. A non-zero β_f, however, implies a non-zero $1 - t_f$, and the 2-core unifies a finite fraction of all vertices of the random graph.

Note that the non-zero solution of Eq. (3.36) sets in continuously at $c(2) = 1$. If we fix a slightly larger $c = 1 + \varepsilon$ with $0 < \varepsilon \ll 1$, we find

$$\frac{\beta_f}{1 + \varepsilon} = 1 - e^{-\beta_f} \, . \tag{3.37}$$

Expanding this equation in β_f and ε we get

$$\beta_f(1 - \varepsilon + \mathcal{O}(\varepsilon^2)) = \beta_f \left(1 - \frac{\beta_f}{2} + \mathcal{O}(\beta_f^2) \right) \tag{3.38}$$

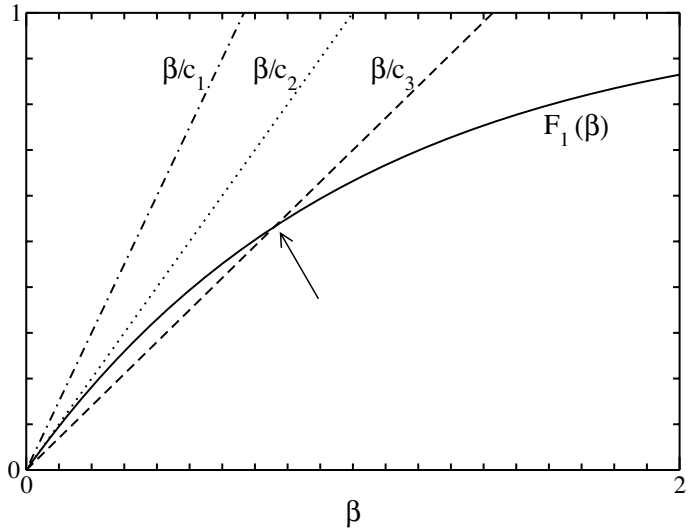


Figure 3.20: Graphical solution of Eq. (3.36): The full line shows $F_1(\beta)$, the straight lines have slopes $1/c_i$ with $c_1 < c_2 = 1 < c_3$. The solution is given by the largest crossing point between the curves. For $c \leq 1$ the only crossing appears in $\beta = 0$. For $c > 1$, a second, positive solution exists. It is marked by the arrow.

and thus $\varepsilon = \beta_f/2$. For the halting time, and thus for the size of the 2-core, we have

$$\begin{aligned} 1 - t_f &= 1 - e^{-\beta_f}(1 + \beta_f) \\ &= \frac{\beta_f^2}{2} + \mathcal{O}(\beta_f^3) \\ &= 2\varepsilon^2 + \mathcal{O}(\varepsilon^3) \, . \end{aligned} \tag{3.39}$$

The critical exponent for 2-core percolation is thus 2, which is in perfect agreement with the numerical findings presented at the beginning of this section.

The case $q = 3$

For $q = 3$, the self-consistency equation for $\beta_f = \beta(t_f)$ becomes

$$\frac{\beta_f}{c} = 1 - e^{-\beta_f}(1 + \beta_f) \, , \tag{3.40}$$

see Fig. 3.21 for the graphical solution. The important difference to the 2-core is that, for $q = 3$, the right-hand side starts with slope zero. A non-zero crossing point developing continuously out of the solution $\beta = 0$ is therefore impossible. In fact, up to $c(3) \simeq 3.35$, no

further crossing point exists, and the existence of a giant component of the random graph is obviously not sufficient for the existence of a giant 3-core. At $c(3) \simeq 3.35$, however, the two curves touch each other at one point $\beta_f > 0$, as marked by the full-line arrow in Fig. 3.21, and *two* new solutions appear. The larger one is, as already discussed, the correct one.

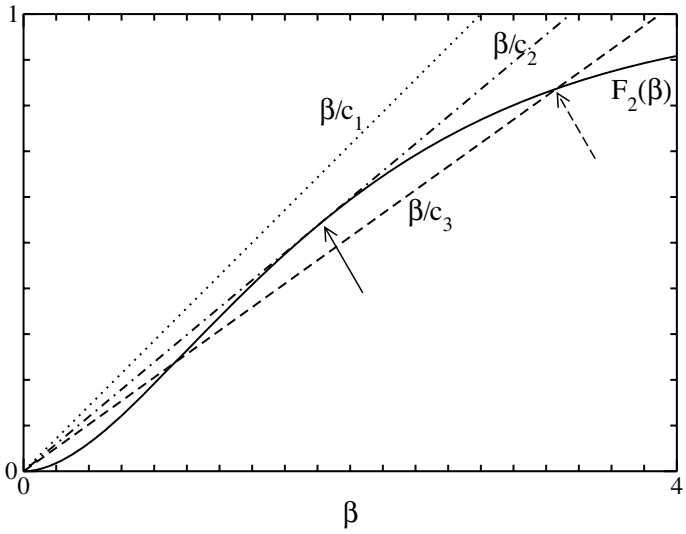


Figure 3.21: Graphical solution of Eq. (3.40): The full line shows $F_2(\beta)$, the straight lines have slopes $1/c_i$ with $c_1 < c_2 = 3.35 < c_3$. The solution is given by the largest crossing point between the curves. For $c \leq 3.35$ the only crossing appears in $\beta = 0$. For $c > 3.35$, two other, positive solutions exist. The correct one is marked by the arrows.

The discontinuous emergence of a new solution in Eq. (3.40) results also in a discontinuous emergence of the 3-core. As observed in the experiments, the latter jumps at $c(3) \simeq 3.35$ from zero size to about 27% of all vertices. This becomes even more pronounced if we look to larger q, e. g., the 4-core appears at $c(4) \simeq 5.14$ and includes $0.43N$ vertices, and the 5-core emerges only at $c(5) \simeq 6.81$ and includes even $0.55N$ vertices.

Results for random-graph coloring

As an immediate consequence we see that almost all graphs with $c < c(q)$ are colorable with q colors, or, formulated differently, that the chromatic number for a graph with $c < c(q)$ is almost surely bounded from above by q. It may, of course, be smaller. We know that the existence of a q-core is not sufficient to make a graph uncolorable with q colors, see Fig. 3.5. As noted in the proof of the theorem on page 35, the reversion of the q-core algorithm can be used to extend the coloring of the q-core to the full graph in linear time. So we see that,

even if the problem is hard in the worst case, almost all graphs of $c < c(q)$ can be efficiently colored with only q colors.

The analysis given by Pittel, Spencer, and Wormald is very similar to an analysis of a simple linear time algorithm that we give in Chap. 8. In general, the probabilistic analysis of algorithms is able to provide useful bounds on properties of almost all random graphs.

Bibliography

[1] L. Euler, Commentarii Academiae Scientiarum Imperialis Petropolitanae **8**, 128 (1736).

[2] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, (Holt, Rinehard and Winston, New York 1976).

[3] K. Appel and W. Haken, Discrete Math. **16**, 179 (1976).

[4] R. L. Graham, M. Grötschel, and L. Lovász, *Handbook of Combinatorics*, (Elsevier Science Publishers, Amsterdam 1995).

[5] L. Lovász and M. D. Plummer, *Matching Theory*, (Elsevier Science Publishers, Amsterdam 1986).

[6] H. Rieger, *Ground state properties of frustrated systems*, in: J. Kertesz and I. Kondor (eds.), *Advances in Computer Simulation*, Lecture Notes in Physics **501**, (Springer, Heidelberg, 1998).

[7] M. Alava, P. Duxbury, C. Moukarzel, and H. Rieger, *Combinatorial optimization and disordered systems*, in: C. Domb and J. L. Lebowitz (eds.), *Phase Transitions and Critical Phenomena*, **18**, 141 (Academic Press, Cambridge 2000).

[8] A. K. Hartmann and H. Rieger, *Optimization Algorithms in Physics*, (Wiley-VCH, Berlin, 2001).

[9] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, (John Wiley & Sons, New York 1998).

[10] B. Korte and J. Vygen, *Combinatorial Optimization – Theory and Algorithms*, (Springer, Heidelberg 2000).

[11] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, Reading (MA) 1974).

[12] R. Sedgewick, *Algorithms in C*, (Addison-Wesley, Reading (MA) 1990).

[13] T. H. Cormen, S. Clifford, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, (MIT Press, Cambridge (USA) 2001).

[14] P. Erdős and A. Rényi, Magyar Tud. Akad. Mat. Kutat Int. Kőzl. **5**, 17 (1960).

[15] B. Bollobás, *Random Graphs*, (Academic Press, New York 1985).

[16] D. Stauffer and A. Aharony, *Introduction to Percolation Theory*, (Taylor & Francis, London 1994).

[17] B. Pittel, J. Spencer, and N. C. Wormald, J. Comb. Theory B **67**, 111 (1996).

[18] N. C. Wormald, Ann. Appl. Prob. **5**, 1217 (1995).

4 Introduction to complexity theory

In this chapter, we present those foundations of theoretical computer science which allow us to describe, analyze and classify combinatorial optimization problems. Basically, one would like to solve optimization problems quickly on a computer. How quickly this can be achieved depends on many circumstances, e. g., the speed of the computer, the quality of the compiler and the sophistication of the coding. These circumstances are within certain limits controllable, e. g., the speed of modern computers is steadily increasing due to technological advances.

On the other hand, there are fundamental limits which cannot be overcome. It is the aim of theoretical computer science to identify these limits and to group combinatorial problems into different classes according to these structural limitations.

We start this chapter with the presentation of Turing machines (TM), which are a simple, although extremely powerful, model of computers. Next we explain Church's thesis that all intuitively computable functions can also be computed by TMs. We furthermore show that there are well-defined functions which cannot be computed by a TM, this is one fundamental limitation one has to be aware of. In the third section, we introduce formal languages and decision problems, which offer the appropriate tools to treat optimization problems within the TM framework. To provide a little background in theoretical computer science, we prove also that there is no universal algorithm, which is able to decide for all programs and all inputs, whether the specific program will halt when run on a given input ("halting problem").

The second half of this chapter is finally dedicated to the previously mentioned classification of combinatorial problems according to their intrinsic hardness, which is measured (in a computer- and implementation-independent way) by the growth of the solution time with the input size of problem instances. Some optimization problems are solvable in polynomial time. They are considered as being *easy*, and are collected in the class P, which is defined in the fifth section. In the subsequent section, this class is embedded into the class NP of possibly harder problems, whose solutions can be verified in polynomial time, but finding them may be exponentially time consuming. The hardest problems to solve in NP are computationally equivalent to each other. This property is described by the term "NP-completeness" which is defined in the seventh section. At the end we prove that the Boolean satisfiability problem is NP-complete, and we add some other, already defined graph-theoretical problems to the list of NP-complete ones.

The very last section of this chapter is dedicated to a short confrontation between the classical concept of worst-case complexity as introduced below, and the more recent alternative concept

of the typical-case complexity which will be dominating the discussion of the further chapters in this book.

4.1 Turing machines

Optimization algorithms are implemented on computers. Modern computers have a very complicated architecture, e. g., the microprocessor uses a pipeline to execute many machine instructions in parallel, floating-point commands are treated by special subunits, the memory is organized in a hierarchical fashion with variable speed of access (registers, cache, random access memory, hard disc). If one tried to analyze how a program is executed on such a computer, one would have a hard time to deal with all these technological details.

For this reason, one uses a very abstract description of a computer, which is reduced to contain only the necessary ingredients. This allows us to analyze programs independently on a specific computer architecture, and enables us therefore to prove fundamental, universal properties. A couple of models for computers have been invented. Here we focus on *Turing machines* (TM), which were first presented in 1936 by Alan Turing [1].

The input to a TM and its output consist of words w, w' over a given alphabet Σ. In this context, we use the following definition:

Definition: $\Sigma^\star$

Let Σ be a finite set of symbols (i. e., an *alphabet*). Then $\Sigma^\star$ is the set of all finite-length sequences of symbols from Σ including the empty word ϵ.

Example: $\Sigma^\star$
Let $\Sigma = \{a, b\}$, then $\Sigma^\star = \{\epsilon, a, b, aa, ab, ba, bb, aaa, \ldots\}$. □

The TM itself consists of a transition function δ, which represents the program code. The lines of the program are called *states*, Q is the set of states. Mathematically, the program is a finite-state machine. A TM has a small memory, which stores its current state. Finally, a TM has a tape, which extends infinitely to the left and to the right. The tape consists of a sequence of tape cells, each tape cell holds exactly one symbol from a tape alphabet Γ, an empty cell holds the special *blank symbol* B. The TM can access the tape via a read/write head, which allows us to access exactly one cell at a time. The TM can move the head stepwise left and right. In Fig. 4.1 a TM is shown.

The formal definition of a TM is as follows:

Definition: *Turing machine (TM)*

A *Turing machine* M is a seven-tuple $M = (Q, \Sigma, \Gamma, B, q_0, \delta, F)$ where

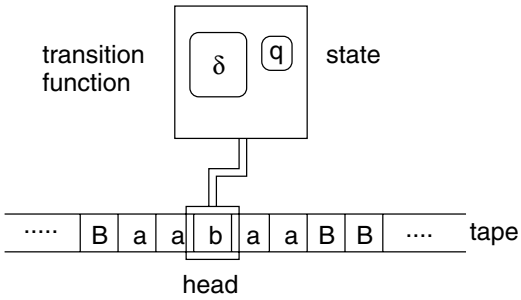


Figure 4.1: A Turing machine.

- Q is a finite set of *states* (internal memory),
- Σ is a finite set of *input symbols*,
- $\Gamma \supset \Sigma$ is a finite set of *tape symbols*,
- $B \in \Gamma \setminus \Sigma$ is the *blank* symbol,
- $q_0 \in Q$ is the *initial* state,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$ is the *transition function* and
- $F \subseteq Q$ is a set of *final* states.

A TM M works as follows. First, the input word is written on the tape surrounded by blanks. Initially M is in state q_0 and the head is on the first symbol of the input word, which extends to the right. Empty tape cells contain the blank symbol B. M executes the program δ iteratively as follows. In each step M first reads the symbol a at the head, then depending on the symbol and the current state q, it obtains $\delta(q, a) = (q', a', D)$. Now, it first writes a new symbol a', then it moves the head to the right ($D = R$), to the left ($D = L$), or leaves it where it is ($D = N$) and finally performs a transition to the new state q'. These steps are iterated, until the new state is in F, then the TM *halts* (i. e., stops running). After halting, the output of the TM is determined as the word beginning at the current position of the head extending right up to (but not including) the first occurrence of a symbol not in Σ.

Since a TM has a very restricted set of commands, writing programs is extremely complicated. Nevertheless, it will turn out that all algorithms, which can be written for any computer, can in principle be implemented on a TM. All functions which can be realized by a TM are called *computable*:

Definition: *computable*

Let M be a TM.

- The function $\Sigma^\star \to \Sigma^\star$ realized by M is denoted by f_M. If M does not halt on input x, we write $f_M(x) =$div.
- A function $f : \Sigma^\star \to \Sigma^\star$ is *computable*iff there is a Turing machine $M^\star$ with $f_{M^\star} = f$.

Note that the functions realized by a computer are defined over words of a finite alphabet. In order to compute mathematical functions $f : \mathbb{N} \to \mathbb{N}$, one has to use a suitable representation of the set $\mathbb{N}$ of natural numbers. One can use, e. g., the unary representation ($\Sigma = \{I\}$, $n \equiv I^n$) or the binary representation ($\Sigma = \{0, 1\}$, $n = \sum_i a_i 2^i \equiv a_n a_{n-1} \dots a_0$, $a_i \in \Sigma$).

Also for mathematical functions, a TM may not halt on a given input x. Functions $f : \mathbb{N} \to \mathbb{N} \cup \{\text{div}\}$ are called *partial* functions.

Many mathematical functions, like the addition of two numbers, require more than one argument, i. e., they are of the form $f : \mathbb{N}^k \to \mathbb{N}$ with some $k > 1$. For this case, the input numbers $(n_1, \dots, n_k)$ are written one after the other on the tape, and they are separated by a symbol $\# \in \Gamma$, i. e., the input takes the form $n_1 \# n_2 \# \dots \# n_k$ on the tape.

In order to understand how a TM works, we will give an example and observe the action of the TM step by step. For this purpose, we still need to describe the current configuration of a TM:

Definition: *configuration*

- A *configuration* of a TM M is a string $\alpha q \beta$ with $\alpha, \beta \in \Gamma^\star$, $q \in Q$. The meaning is as follows. The content of the tape is $\alpha\beta$ surrounded by blanks B, the head is positioned at the first letter of β, and M is in state q. Note also that α and β may contain the blank symbol B. Outside $\alpha\beta$, there are only blanks.
- If configuration $\alpha' q' \beta'$ follows directly after configuration $\alpha q \beta$ when running M, we write $\alpha q \beta \vdash \alpha' q' \beta'$.

Using this definition, we finally arrive at our first Turing program, the subtraction of two numbers.

Example: Subtraction of numbers

For simplicity, we use the unary representation. The function has two arguments $x, y > 0$, i. e., the initial state is $\epsilon q_0 x \# y$. The TM shall return $x - y$. We assume $x \geq y$, which allows us to keep the program short.

The basic idea of the computation is to use the equality

$$x - y = \begin{cases} x & \text{if} \quad y = 0 \\ (x-1) - (y-1) & \text{else.} \end{cases} \tag{4.1}$$

Hence, the TM removes iteratively one I symbol from both the x and the y strings. Symbols from x are overwritten from the left with B, symbols from y from the left with $\#$. The main work of the TM consists in moving the head back and forth between x and y, each time overwriting exactly one symbol. If the remaining part of y is empty, e. g., it is reduced to zero, the tape space behind the remaining word

x is filled with blanks, then the TM moves the head to the first remaining letter of x and finally M stops. Formally, the TM is given as follows:

- $Q = \{q_0, q_1, \ldots, q_6\}$
- $\Sigma = \{I, \#\}, \Gamma = \{I, \#, B\}$
- $F = \{q_6\}$
- δ given by the following table. For each state $q \in Q$ (column 1), there is one line. In columns 2–4 of the table, the values $\delta(q, a)$ for the three possibilities of symbols $a \in \Gamma$ under the head are given. The last column describes the meaning of the given state in words.

	I	$\#$	B	remark
q_0	q_1BR	q_6BN		remove I at left of x, i. e., build $x-1$
q_1	q_1IR	$q_2\#R$		move right to first $\#$
q_2	$q_3\#R$	$q_2\#R$		move right to y and build $y-1$
q_3	q_4IL		q_5BL	check whether $y = 0$ (i. e., empty)
q_4	q_4IL	$q_4\#L$	q_0BR	move left up to beginning of x
q_5	q_5IL	q_5BL	q_6BR	same as q_4 but erase all $\#$
q_6				finished

Empty fields ($q \neq q_6$) in the table, mean that the configuration is not changed ($\delta(q, a) = (q, a, N)$). This would result in an infinite loop in principle, but δ is defined here in a way that these configurations can never occur.

As an example for the execution of the program δ, we consider the inputs $x = 3, y = 2$. Then we get the following sequence of configurations:

$$\begin{aligned}
\epsilon q_0 III\#II &\vdash \epsilon q_1 II\#II \vdash Iq_1I\#II \vdash IIq_1\#II \\
&\vdash II\#q_2II \\
&\vdash II\#\#q_3I \\
&\vdash II\#q_4\#I \vdash IIq_4\#\#I \vdash Iq_4I\#\#I \vdash \epsilon q_4II\#\#I \vdash \epsilon q_4BII\#\#I \\
&\vdash \epsilon q_0II\#\#I \\
&\vdash \epsilon q_1I\#\#I \vdash Iq_1\#\#I \\
&\vdash I\#q_2\#I \vdash I\#\#q_2I \\
&\vdash I\#\#\#q_3\epsilon \\
&\vdash I\#\#q_5\# \vdash I\#q_5\# \vdash Iq_5\# \vdash \epsilon q_5I \vdash \epsilon q_5BI \\
&\vdash \epsilon q_6I
\end{aligned}$$

□

4.2 Church's thesis

As already mentioned, there are other models for computation, but here we only mention two prominent examples.

- A *register machine* resembles a realistic computer more than does a TM. It contains several registers to store numbers and has built-in operations like addition and multiplication. Similar to TM, there are internal states and transitions between states.
- Mathematically, functions can be defined recursively. An extension allows for breaking the recurrence, leading to so-called μ-recursive functions.

Over the years, theoretical computer science [2] has proved, that that register machines, μ-recursive functions and TMs are computationally equivalent. This means that each function, which can be defined or computed using one model, can be computed with the other models as well.

The running time of a program is measured in terms of steps of the machine on which it is executed. The running time may depend on the machine model which is applied. This is true also for the asymptotic running time, e. g., a program which runs in $\mathcal{O}(n^2)$ on a TM may run in $\mathcal{O}(n)$ on a register machine. The reason is that a register machine can access different memory cells in constant time, while a TM first has to move the head to the cell which is addressed. The important fact is that a program which runs in polynomial time on one machine, i. e., it is considered to be fast, runs also polynomially on any other meaningful machine model.

So far, nobody has given a applicable recipe for the computation of a function, which cannot be realized by a TM. This leads to:

Church's thesis: *The class of computable functions is equal to the class of intuitive computable functions.*

The implication ($\rightarrow$) is clear. Each program δ of a TM can be executed step by step (e. g., by hand), hence it is intuitive computable.

($\leftarrow$) Since there is no definition of "intuitive computable", it is impossible to "prove" this direction of the equivalence. But so far nobody has found a counterexample.

Nevertheless, there are functions, which are *not computable*. This may sound a little bit confusing in the beginning, since we said above that all intuitively computable functions are computable. The solution is that the non-computable functions are also intuitively not computable, as we will see below.

To prove our statement, we first introduce a correspondence between TMs and natural numbers. To achieve this, we assume from now on, without loss of generality, $\Sigma = \{0, 1\}$. We can do this, because we can code any finite alphabet by finite-length words over $\{0, 1\}$ (as any natural number can be coded by a binary string).

For the same reason, we are restricting ourself to $\Gamma = \Sigma \cup \{B\}$. Therefore, Γ contains three symbols $X_1 \equiv 0$, $X_2 \equiv 1$ and $X_3 \equiv B$. The finite set of states is denoted by $Q \equiv \{q_i\} \equiv \{q_1, q_2, \ldots, q_n\}$. The three possible directions of the head movement

are denoted by $D_1 \equiv L$, $D_2 \equiv R$ and $D_3 \equiv N$. Now each TM is just given by the $\alpha = 1, \ldots, 3n$ possible results $\delta(q_1, X_1)$, $\delta(q_1, X_2)$, $\delta(q_1, X_3)$, $\delta(q_2, X_1), \ldots, \delta(q_n, X_3)$. If for result α we have $\delta(q_i, X_j) = (q_k, X_l, D_m)$ we assign a string of zeros and ones of the form $\text{code}(\alpha) = 10^k 10^l 10^m$. Hence, we can describe the TM M completely by the string $\langle M \rangle \equiv \text{code}(1)\text{code}(2)\ldots\text{code}(3n)$, which corresponds to a natural number via the binary representation. Hence, for any computable function, we have a TM, and for each TM, we have a unique natural number n representing it, we denote the corresponding TM by M_n. On the other hand, we can interpret any natural number n as a binary string (no leading zeroes). Sometimes this string gives a meaningful TM M_n as described above, the function realized by M we call f_n. If the resulting string does not obey the described syntax, we assign this number a TM which just performs infinite loops on all inputs, i. e., we have $f_n(x) = \text{div}$ for all inputs x and f_n is computable.

We remark that the generation of f_n can be computed as well. It is possible to construct a TM U, which gets the input $\langle M \rangle \# x$, then U constructs the program δ of M by decoding the corresponding string $\langle M \rangle$ and then interprets the program on input x. This TM is called the *universal TM*, because it can realize all computable functions f_n.

To summarize, we have defined a computable function f_n for each natural number n and we can assign to each TM (and to its function f_M) a natural number, hence we are sure that all computable functions appear in the list $(f_1, f_2, f_3, \ldots)$

We now use this construction to perform our proof that there are functions which are not computable. The basic idea is to use the principle of *diagonalization*. This principle was introduced by Georg Cantor to show $\mathbb{R} \not\subset \mathbb{Q}$. It works by constructing a table of infinite size, were each row corresponds to one function f_n and in each column i, the results of the functions on input x, are written, see Fig. 4.2. The idea of the proof is to define a function, which differs on the diagonal from each computable function.

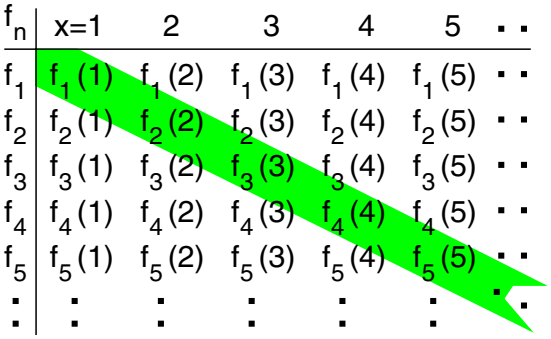


Figure 4.2: Principle of diagonalization: define a function which differs from all computable functions on the diagonal.

Proof:

We define the following function

$$f^*(x) = \begin{cases} 1 & \text{for} \quad f_x(x) = \text{div} \\ \text{div} & \text{else} \end{cases} \tag{4.2}$$

Evidently, this is a well defined partial function over the natural numbers. The point is that it is different from all computable functions f_n, i. e., f^* itself *cannot* be computable:

$$\forall n : f_n(n) \neq f^*(n), \quad \Rightarrow \quad \forall n : f_n \neq f \tag{4.3}$$

QED

Please note that Eq. (4.2) does *not* provide a way to compute $f^\star$. One could try it as follows. Using the universal TM, one can run the xth TM on input x. If the TM stops after a while, one can start an infinite loop. In this case $f^*(x) =$div is correctly computed. But if $f_x(x)$ does not stop, one does not know! One knows just that f_x keeps on running. It might stop in 10 minutes, or, if not, it might stop in 10^{100} years or it might never stop. Hence, one will never know what is the correct result for $f^*(x)$ in this case. Below we will formally prove, again by using the principle of diagonalization, that the *halting problem* cannot be solved. It means that it is impossible to invent an algorithm, which is able to decide for all programs and all inputs, whether the program will halt when run on the given input.

4.3 Languages

So far, when solving optimization problems, we look for the minimum or maximum of some given function and return the value and possibly also the corresponding argument. It is true that we could write a TM finding the minimum of a given function. However, to make fundamental statements in this field, such as showing that the computational "hardness" of two problems is equal, it is easier to treat pure symbolic problems. To make a link between combinatorial optimization problems and a certain class of symbolic problems, which is defined below, we first introduce languages.

Definition: *Language*

Let Σ be a finite alphabet. A *language* L is a subset $L \subset \Sigma^\star$.

We can use TMs to define languages. We assume that $0, 1 \in \Sigma$. We say for $1 \in \Sigma$ that a TM M *accepts* input x, iff $f_M(x) = 1$, i. e., an input is not accepted if M does not halt on input x, or if M halts but the result is not equal to 1. Hence, each TM represents a language, which consists of all words accepted by the TM. This principle of defining languages is used in the following definition.

Definition: *Decidable, semi-decidable*

Let L be a language.

- L is called *decidable*, if there is a TM M which stops on *all* inputs and accepts an input x iff $x \in L$.
- L is called *semi-decidable*, if there is a TM M which accepts an input x iff $x \in L$.

For a decidable language L, you always get an answer from the corresponding TM: "yes, it belongs to L", or "no, it does not belong to L". For a semi-decidable language L, you get also the answer "yes" if $x \in L$, but $x \notin L$ maybe unknown because the TM may never stop on input x. In fact, each decidable language is also semi-decidable. We next give an example of a decidable language.

Example: Decidable

For $\Sigma = \{0, 1\}$ the language $L = \{1^n | n \geq 0\}$ is decidable.

Proof: The following TM decides L.

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, B\}$
- $F = \{q_4\}$
- δ given by:

	0	1	B	remark
q_0	q_2BN	q_0BR	q_1BR	read input string up to 1st 0 or end
q_1			$q_3 1R$	write answer 1
q_2			$q_3 0R$	write answer 0
q_3	q_4BL	q_4BL	q_4BL	write final blank
q_4				finished

□

Definition: *Decision problem*

A *decision problem* is a question for which the answer is either "yes" or "no".

Each instance of a combinatorial optimization problem (OPT) "What is the Minimum of $H(\underline{\sigma})$?" is equivalent to the decision problem (DEC(K)) "Is there a $\underline{\sigma_0}$ such that $H(\underline{\sigma_0}) \leq K$".

Proof:

($\rightarrow$) Assume that (OPT) is solvable (i. e., we have a computer program, which calculates the optimum). Given K, then one can compute $z \equiv \min(H)$. Now, if $z \leq K$, then answer "yes", else answer "no".

($\leftarrow$) Assume that (DEC(K)) is solvable. Since the problem is combinatorial, the set of possible values $H(\underline{\sigma})$ is discrete (e. g., for TSP: without loss of generality, we can assume that all distances are from $\mathbb{N}$, e. g., measured in a very small unit like cm). Hence the possible outcomes are discrete values as well. One now can identify all possible values of the solutions $\ldots < K_{-2} < K_{-1} < K_0 < K_1 < K_2 < \ldots$.

The most simple algorithm to realize (OPT) works as follows: Set $i = 0$. If DEC(K_i)="yes" set $i := i - 1$ else $i := i + 1$. Repeat until the answer changes. (A faster approach is to use iterated interval bisection.) QED

Now we have mapped an optimization problem onto a decision problem. Furthermore, each decision problem can be formulated as a language L: L just contains all the suitably coded instances, for which the answer is "yes". We can code the "yes" instances with a similar scheme like we have used to code TMs. For example, for the TSP, the coding string could be $n\#a_{11}\#a_{12}\#a_{12}\#\ldots\#a_{nn}$ where n is a binary representation of the number of cities, and $(a_{ij})_{ij}$ is a matrix with all distances, which is coded line by line.

We have already seen that languages can be easily represented by TMs, hence now we have a special representation of combinatorial optimization problems by TMs. For all optimization problems, we can find the optimum (maybe after a long time). Hence, the decision versions of optimization problems considered here are decidable, i. e., a special subset of decidable languages.

Before we move on to a classification of optimization problems (decision versions) in the next chapter, and before we show how one can map different problems onto each other, we show that the *halting problem* is *not* decidable.

4.4 The halting problem

The halting problem is fundamental in theoretical computer science. We show that there is no universal algorithm, which is able to decide for all programs and all inputs, whether the program will halt when run on the input. It would be very desirable to have such an algorithm, because it would save a lot of debugging time spent on finding infinite loops. But, as we will prove below, there is no hope that there is such an algorithm. Thus, one can save resources and concentrate on solvable problems.

In order to perform the proof, we start with the definition of the set D, which contains the numbers n as binary strings, where the nth TM does not accept the input n, i. e., the number representing itself.

Definition: D

Let w_n be the nth word in the canonical order over $\{0,1\}^\star$ and M_n the nth TM (see above). Then:

$D \equiv \{w_n | M_n \text{ does not accept } w_n\}$.

Theorem: *D is not decidable.*

Proof:

We perform a proof by contradiction. Assume that D is decidable. Then we know by definition of "decidable" that there is a TM M_j which halts on all inputs and accepts exactly the members of D $(*)$. Now we have two cases:

Assume $w_j \in D \quad \overset{(*)}{\Rightarrow} \quad M_j$ halts on input w_j. Contradiction to Def. of D

Assume $w_j \notin D \quad \overset{(*)}{\Rightarrow} \quad M_j$ does not accept w_j. Contradiction to Def. of D. QED

Furthermore, the complement $\overline{D} = \{w_i | M_i \text{ accepts } w_i\}$ is also not decidable. If it were so, one could transform a TM which decides $\overline{D}$ easily into a TM, which decides D.

Now we can proceed with the definition of the halting problem.

Definition: *Halting problem H*

Let bet $\langle M \rangle$ be the code word $\in \{0,1\}^\star$ describing TM M.

$H \equiv \{\langle M \rangle w | M \text{ halts on input } w\}$. This means H consists of the string describing TMs and inputs to the TMs, where the TM halts on the given input.

Theorem: *H is not decidable.*

Proof:

We perform the proof by contradiction. We assume that H is decidable, hence there is a TM M halting on all inputs accepting exactly H.

Then the the following algorithm ALG (i. e., a TM) halts on all inputs and computes $\overline{D}$. This is an obvious contradiction to the non-decidability of $\overline{D}$ proved before.

The algorithm ALG is given as follows, let $w \in \Sigma^*$:

algorithm ALG(w)
begin
  Calculate n such that w is the nth word in $\{0,1\}^\star$
  Obtain $\langle M_n \rangle$
  Run M with input $\langle M_n \rangle w_n$
  if M answers "no" **then**
    return: $w \notin D$
  if M answers "yes" **then**
  begin
    run M_n with w_n (it will halt)
    if M_i accepts w_n **then**
      return: $w \in \overline{D}$
    else
      return: $w \notin \overline{D}$
  end
end

QED

This does *not* mean that one cannot prove for some *given* program whether it halts on a given input. This is possible for (say) small programs. The above statement means just that there is no algorithm suitable for *all* programs. Please note that H is semi-decidable. If a program halts on a given input, then you can prove it just by running the program with the input.

The situation is similar in mathematics. One *can* find proofs of mathematical theorems, but there is no algorithm to generate all proofs. At this point one must rely on the capabilities of the mathematicians.

There are other similar fundamental problems which are not decidable. The most important is probably $L(f) \equiv \{\langle M \rangle | f_M = f\}$, i. e., the set of all TMs which realize a given function. Since $L(f)$ is not decidable, there is no way to obtain an "automatic verification" of programs, i. e., the process of proving the correctness of a program is a hard task, or, in other words, "there is no program without a bug".

4.5 Class P

We now define the class of problems, where the running time is bounded by a polynomial in the size n of the input. The size of the input of a TM is just the number n of characters of the input string. Note that for practical considerations more convenient measures for the size of the input are usually chosen, e. g., the number of cities for the TSP.

Definition: *Class P*

P is the class of (decision) problems solved by a TM where the worst-case running time $t_M(n)$, i. e., the maximum over all running times with input size n, is bounded by a polynomial in n.

Here, we will only consider the decision variants of the optimization problem, hence we regard P as a subset of all decidable problems. In general there are many polynomial solvable problems. We now give some examples.

- As we have already seen in Sec. 2.3, the multiplication of numbers of size n can be solved in time $\mathcal{O}(n^{\log_2 3})$ by a divide-and-conquer approach. This is bounded by $\mathcal{O}(n^2)$, the time complexity of the naive algorithm.
- The sorting of a dataset with n elements, e. g., when printing a library catalog including n books in alphabetical order, can be performed in $\mathcal{O}(n \log n)$, which is also bounded by the polynomial $p(n) = n^2$.
- In the chapter on graph theory, in Sec. 3.1.1, we have shown that the question "Has a given graph an Euler circuit?" can be solved by iterating over all vertices (and their neighbors), i. e., in a time bounded by $\mathcal{O}(n^2)$ if n is the number of vertices.
- Another example from graph theory is: given a graph with distance labels on the edges, what is the shortest path between two selected cities? This problem can be solved in, at most, quadratic time in the number of cities. The algorithm will be presented in Sec. 11.4.
- A problem from physics, which we will study in greater detail later on, see Sec. 11.7, is the ground state calculation of planar spin glasses, which can be solved in polynomial time using a matching algorithm.

On the other hand, there are problems which are definitely not polynomial.

- Non-decidable problems like the halting problem are not polynomial, since for some instances the corresponding TM does not halt at all, i. e., the running time cannot be bounded by a polynomial.
- Many optimization problems, like the ground-state calculation of spin glasses with discrete couplings, have an exponential degeneracy, i. e., a non-zero entropy per spin. Hence, the calculation of *all* ground states needs an exponential output size, which takes an exponential time to write.
- There are few problems with small output size, but a proven non-diverging running time which increases exponentially or faster with the problem size.

  One example is the arithmetics which has only $+/-$ as operations and $=$ as relation. It was shown to be decidable by by M. Presburger [3]. The decision whether a given formula follows from given axioms takes $\mathcal{O}(2^{2^N})$, as shown by by M. J. Fischer and M. O. Rabin [4].

The axioms defining the arithmetic are:

1. $\forall x: \quad \neg(0 = x + 1)$.
2. $\forall x \forall y: \quad \neg(x = y) \Rightarrow \neg(x + 1 = y + 1)$.
3. $\forall x: \quad x + 0 = x$.
4. $\forall x \forall y: \quad (x + y) + 1 = x + (y + 1)$.
5. This is an axiom scheme consisting of infinitely many axioms. If $P(x)$ is any formula involving the constants 0, 1, $+$, $=$ and a single free variable x, then the following formula is an axiom: $(P(0) \wedge \forall x: P(x) \Rightarrow P(x + 1)) \Rightarrow \forall x: P(x)$.

Concepts such as divisibility or prime numbers cannot be formalized in Presburger arithmetic. Here is a typical theorem that can be proved from the above axioms [5]:

$\forall x \forall y: \quad ((\exists z: x + z = y + 1) \Rightarrow (\forall z: \neg(((1 + y) + 1) + z = x)))$.

It says that if $y + 1 \geq x$, then $y + 2 > x$.

For may interesting optimization problems it is unknown whether they are in P or not. A special class called NP has been introduced for them, which is defined in the next section.

4.6 Class NP

For hard optimization problems, *finding* the optimum or answering the question "Is there a solution with $H(\underline{\sigma}) \leq K$?" takes a long running time with current algorithms. On the other hand, for many problems defined before, it is easy to *verify* if some given candidate solution $\underline{\sigma}$ fulfils the constraint $H(\underline{\sigma}) \leq K$. Taking TSP as an example, it is easy to calculate the length $L(\underline{\sigma})$ of a given tour; one has simply to add up the distances. After this one can check whether $L(\underline{\sigma}) \leq K$ or not.

An open question remains. How can we find a good solution in between the huge number of candidates? For a TSP with n towns, there are $(n - 1)!$ possible round tours, such that checking one after the other is not possible in polynomially bounded time.

Instead of answering this question, we just avoid it. We assume that a tentative solution is just available, e. g., it could have been proposed by an *oracle*. To formalize the concept of oracles, *non-deterministic* TMs have been introduced in theoretical computer science. This allows us, e. g., to formally perform proofs like showing that two such problems are equivalent concerning their computational complexity.

Definition: *Non-deterministic Turing machine (NTM)*

NTMs are defined as (deterministic) TMs, with the only exception that δ is now a *relation* on $(Q \times \Gamma) \times (Q \times \Gamma \times \{L, R, N\})$ instead of a function.

An NTM chooses at each step *one* transition among all possible transitions. This means each (deterministic) TM is an NTM as well. We say that an NTM *accepts* an $w \in \Sigma$ if there *exists* at least one sequence of configurations leading to the output 1.

To obtain the running time of an NTM with given input w one has to consider *all* possible sequences of transitions on w and to take the minimum running time among them. The worst-case running time $t_M(n)$ is again the maximum of running times over all inputs w with length n.

Definition: *Worst-case running time of NTMs*

$$t_M(n) \equiv \max_{w\in\Sigma,|w|=n} \min_{\vdash}(\#\text{steps}) \,.$$

Using a TM, we have defined a language L as a set of words, which are accepted by the TM. In the same way, languages can be defined for NTM. The difference is now that a word w belongs to L if there is at least *one* sequence of configurations leading to acceptance. Other possible sequences of configurations with input w are allowed to finish in a non-accepting configuration.

Definition: *Class NP*

The class NP of *non-deterministic polynomial* problems contains all decision problems L where there exists an NTM M accepting L and $t_M(n)$ is bounded by a polynomial $p(n)$.

Example: TSP∈NP

It is easy to show that the TSP∈ NP. A possible NTM M accepting TSP looks as follows:

algorithm TSP-NTM($G, \{d_{ij}\}, K$)
begin
  generate tour non-deterministically and write it on tape (in linear time)
  calculate (deterministically) the length L of tour (in polynomial time)
  if $L \leq K$ **then**
    write 1 on tape
  else
    write 0 on tape
end

□

Unfortunately, it is hard to *build* an NTM in reality. On the other hand, one can easily *simulate* an NTM using a deterministic TM. The basic idea is that the TM iteratively tries all possible sequences of calculations of the NTM. If one of them is accepted, the input word is accepted by the TM, or else it is not accepted. The running time of this TM is obviously exponential, as stated by the following theorem.

Theorem: *For all $L \in$ NP there exists a polynomial p and a TM M such that M accepts L in running time $2^{p(n)}$.*

Proof:

Let L be in NP. By definition, there exists an NTM M' accepting L in $\mathcal{O}(q(n))$, q being a polynomial. We set $k \equiv 3|Q||\Gamma|$. This is an upper bound for the number of choices at each step of the NTM. Then the following TM M accepts L deterministically:

algorithm $M(w)$
begin
  calculate $m = q(|w|)$
  for all m-tuples $\underline{i} \equiv (i_1, \ldots, i_m) \in \{1, \ldots, k\}^m$ **do**
  begin
    write $\underline{i}$ on tape (unused space, leftmost of input)
    simulate $M'(w)$, picking transitions according $\underline{i}$:
      if transition is not defined **then**
        continue loop
    if M' accepts w **then**
      output: 1 and exit
  end
  output: 0
end

It is clear that the NTM M' accepts w if and only if TM M accepts w.

The main loop over all possible sequences of configurations takes k^m steps. Hence, the running time of the TM is, within a polynomial factor, proportional to $k^m m = 2^{q(n)\log_2 k} q(n)$ which is bounded by $2^{p(n)}$, $p(n)$ being a polynomial. QED

Since the running time is exponential, i. e., finite, for each language $L \in$NP, the corresponding TM halts on all inputs. Therefore, each $L \in$ NP is decidable. Furthermore, since a TM is also an NTM we have P$\subset$NP. In the next section, we introduce the concept of NP-completeness, which describes an equivalence class of decision problems and shows that the most common hard optimization problems are computationally equivalent.

4.7 Definition of NP-completeness

To show this type of computational equivalence between different combinatorial problems, we need a concept of transformation between different problems. Since we are ultimately interested in polynomial algorithms, this transformation must not include more than polynomially many steps. This requirement leads to the following definition.

Definition: *Polynomially reducible*

Let L_1, L_2 be languages over Σ_1, Σ_2.
L_1 is called *polynomially reducible* to L_2 (we write $L_1 \leq_p L_2$) if there is a function $f : \Sigma_1^\star \to \Sigma_2^\star$, which can be calculated by a polynomial time TM, such that

$$w \in L_1 \quad \Leftrightarrow \quad f(w) \in L_2 \,.$$

This means that, for two problems $L_1 \leq_p L_2$, an algorithm deciding L_2 can be polynomially translated into an algorithm deciding L_1. The function f maps all words in L_1 into words in L_2, and any word not in L_1 is mapped onto a word not being element of L_2. To decide whether some w is in L_1, we can therefore first apply f and than the algorithm deciding L_2, see Fig. 4.3.

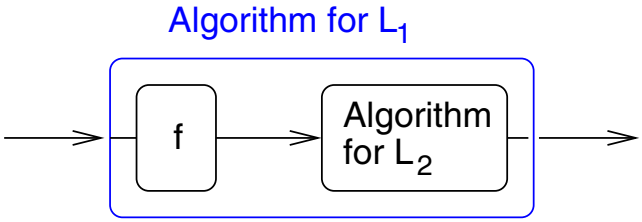


Figure 4.3: Polynomial-time reducibility: an algorithm deciding L_1 consisting of the transformation f and the algorithm for deciding L_2.

As an example, we show that the Hamiltonian cycle problem is polynomially reducible to the traveling-salesman problem. We are interested in knowing whether G has a Hamiltonian cycle, i. e., whether it is possible to visit each vertex exactly once in a cycle. Formally, graphs have to be coded by words over a suitably chosen alphabet Σ_1 and HC contains all words, which represent graphs having a Hamiltonian cycle. We would than have to look for a function $f : \Sigma_1^\star \to \Sigma_2^\star$ which maps words representing (unlabeled) graphs onto words representing graphs which have distance labels.

Since we know that it is, in principle, possible to map graphs onto words over a suitably chosen alphabet, we can concentrate on the essence. This means that we describe just a transformation between unlabeled graphs and graphs having distance labels to show HC$\leq_p$TSP. In the same informal manner, we will later describe other transformations between different decision problems, without returning to the language formalism.

Example: HC$\leq_p$TSP
Given a Graph $G = (V, E)$, with $N = |V|$. The transformation function f creates the complete graph $K_N = (V, V^2)$ and assigns the distance matrix $a_{ij} = 1$ if $\{i, j\} \in E$ and $a_{ij} = 2$ else. We consider TSP(N), i. e., ask whether a round trip of length N exists. Obviously f can be realized in polynomial time.

We have $G \in$ HC
$\Leftrightarrow \exists$ cycle through all nodes in G
$\Leftrightarrow \exists$ cycle in K_N of size N using only edges with $a_{ij} = 1$
$\Leftrightarrow \exists$ round trip in K_N of total distance $\leq N$
$\Leftrightarrow K_N \in$ TSP(N). □

Polynomial reducibility is transitive. Later, this property will be crucial for constructing chains of polynomially reducible problems, which forms the main step in proving that all interesting hard optimization problems are equivalent.

Theorem: $(L_1 \leq_p L_2$ *and* $L_2 \leq_p L_3) \Rightarrow L_1 \leq_p L_3$.

Proof:

Let L_1, L_2, L_3 be languages over alphabets $\Sigma_1, \Sigma_2, \Sigma_3$. By definition of $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ we have functions:

$f : \Sigma_1^\star \to \Sigma_2^\star$, f computable in polynomial time and $w \in L_1 \Leftrightarrow f(w) \in L_2$

$g : \Sigma_2^\star \to \Sigma_3^\star$, g computable in polynomial time and $w' \in L_2 \Leftrightarrow g(w') \in L_3$

Hence, we get: $w \in L_2 \Leftrightarrow g(f(w)) \in L_3$ and $g \circ f$ is computable in polynomial time. This means $L_1 \leq_p L_3$. QED

A decision problem (i. e., a language) L is at least as hard as any other problem in NP, if all NP problems can be mapped onto it, i. e., if all languages $L' \in$NP are polynomially reducible to L. This leads to the following definitions:

Definition: *NP-hard, NP-complete*

- A language L is called *NP-hard* if for all $L' \in$NP we have $L' \leq_p L$.
- A language L is called *NP-complete* if it is NP-hard and $L \in NP$.

This definitions seems rather restrictive, but we will show below that NP-complete problems indeed exist. Before however, we want to underline the crucial importance of this concept. One should keep in mind that, up to now, *no* polynomial-time algorithm for one of these problems has been found, i. e., all known algorithms have an exponential worst-case running

time. If, however, someone came up with *one* polynomial algorithm for *one* NP-complete problem, then *all* problems in NP would be solvable in polynomial running time using the construction of Fig. 4.3. In this sense, the NP-complete problems are the hardest problems in NP, and they are equivalent to each other with respect to the existence (or non-existence) of polynomial decision algorithms. We thus conclude

Theorem: *Let L be NP-complete. If $L \in$P then P=NP.*

Proof:

Let L be NP-complete and $L \in$ P. We have already shown P$\subset$NP before, hence we have to show NP$\subset$P to obtain P=NP, i. e., we have to show that $L' \in$ NP implies $L' \in$ P:

Let $L' \in$ NP. Since L is NP-complete, we know $L' \leq_p L$ by definition. Thus, there is a polynomial time TM M with $w \in L' \Leftrightarrow f_M(w) \in L$. Since $L \in$ P, there is a polynomial time TM M' with $w' \in L \Leftrightarrow f_{M'}(w') = 1$. Combined we have $w \in L' \Leftrightarrow f_{M'}(f_M(w)) = 1$. Since $f_{M'} \circ f_M$ works in polynomial time, we have $L' \in$ P. QED

So far, we have introduced the concept of NP-completeness, and we have discussed its importance. But are there any NP-complete problems at all? We will now introduce the satisfiability problem (SAT), which is the first one which was identified to be NP-complete [6]. SAT is not defined over graphs, as HC and TSP, but it is defined over Boolean formulas.

Definition: *Satisfiability (SAT), K-SAT*

A *Boolean variable* x_i may only take the values 0 (false) and 1 (true). Here we consider three *Boolean operations*:

- NOT $\overline{\quad}$ (*negation*): the *clause* $\overline{x_i}$ ("NOT x_i") is true ($\overline{x_i} = 1$), if and only if x_i is false: $x_i = 0$
- AND $\wedge$ (*conjunction*): the *clause* $x_i \wedge x_j$ ("x_i AND x_j") is true, iff both variables are true: $x_i = 1$ AND $x_j = 1$.
- OR $\vee$ (*disjunction*): the *clause* $x_i \vee x_j$ ("x_i OR x_j") is true, iff at least one of the variables is true: $x_i = 1$ OR $x_j = 1$

The three logical operators NOT ($\overline{x}$), AND ($\wedge$) and OR ($\vee$) can also be represented by so-called *truth tables*:

x_1	$\overline{x}_1$
0	1
1	0

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

(4.4)

A variable x_i and its negation $\overline{x_i}$ are called *literals*. Using parentheses, different clauses may be combined to produce complex *Boolean formulas*, e. g., $(x_1 \vee \overline{x}_2) \wedge (x_3 \vee x_2)$.

A formula is called *satisfiable*, if there is at least one assignment for the values of its variables such that the whole formula becomes true. For example, the formula $(\overline{x_1} \vee x_2) \wedge \overline{x_2}$ is satisfiable, because for $x_1 = 0$, $x_2 = 0$, it is true. The formula $(\overline{x_1} \vee x_2) \wedge \overline{x_2} \wedge x_1$ is not satisfiable, because $\overline{x_2} \wedge x_1$ implies $x_1 = 1, x_2 = 0$, but then $(\overline{x_1} \vee x_2)$ is false.

***Satisfiability*:** (SAT) For a given formula F, is there a satisfying assignment?

For the K-SAT problem, formulas of the following type are considered, called K-CNF (*conjunctive normal form*) formulae: each formula F consists of m clauses C_p combined by the AND operator:

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m. \tag{4.5}$$

Each clause C_p consists of exactly K literals l_{pq} combined by the OR operator:

$$C_p = l_{p1} \vee l_{p2} \vee \ldots \vee l_{pk}. \tag{4.6}$$

The class K-SAT consists of all problems of the form "is F satisfiable?" where F is a K-CNF formula.

One can also consider formulas in CNF, i. e., those consisting of a conjunction of clauses, where each clause is a disjunction of an arbitrary finite number of literals. Hence, different clauses may have a different number of literals. Note that every Boolean formula can be rewritten in CNF. The class SAT thus consists of all problems of the form "Is F satisfiable?" where F is a CNF formula.

We have already seen that some decision problems are undecidable. For these problems it has been proven that no algorithm can be provided to solve it. The K-SAT problem on the other hand is decidable. The simplest algorithm uses the fact that each formula contains a finite number n of variables. Therefore, there are exactly 2^n different assignments for the values of all variables. To check whether a formula is satisfiable, one can scan through all possible assignments and check whether the formula evaluates to true or to false. If for one of them the formula is true, then it is satisfiable, otherwise not. This algorithm obviously runs in time $\mathcal{O}(2^n)$

Example: K-SAT

For example $(x_1 \vee x_2) \wedge (\overline{x_4} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$ is a 2-CNF formula, while $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4})$ is a 3-CNF formula.

In the Table 4.1 all possible assignments for the variables of $(x_2 \vee x_3) \wedge (x_1 \vee \overline{x_3})$ and the results for both clauses and the whole formula are displayed as a truth table.

Table 4.1: Truth table.

x_1	x_2	x_3	$x_2 \vee x_3$	$x_1 \vee \overline{x_3}$	$(x_2 \vee x_3) \wedge (x_1 \vee \overline{x_3})$
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Since there is a variable assignment which evaluates to 1, the formula $(x_2 \vee x_3) \wedge (x_1 \vee \overline{x_3})$ is satisfiable. □

Furthermore, checking whether a given assignment satisfies a K-CNF formula can be performed in linear (i. e., polynomial) time, by just running through all clauses from left to right until an unsatisfied clause is found or until all clauses are treated. Thus, we have K-SAT $\in$ NP.

We will now show that SAT is NP-complete. This means, for *any* given problem $L \in$ NP, we must provide a polynomial-time transformation to SAT formulas such that for each member of L the corresponding formula is satisfiable and for all other words the corresponding formula is not satisfiable. Since we do not know anything about L, we cannot give a transformation! The way out of this problem is given by the fact that there is an NTM M which decides $w \in L$. The basic idea of the proof for the NP-completeness of SAT is to construct a SAT formula that is satisfiable if and only if there is an accepting sequence of configurations of M when run with input w, which is, by definition, equivalent to $w \in L$.

As a preparation of the proof, we first construct for each NTM M which runs in polynomial time $\mathcal{O}(q(n))$ an equivalent NTM M' (running also in polynomial time) where each computation consists of two phases, a non-deterministic phase and a purely deterministic phase:

1st phase: write separator "#" and subsequently write non-deterministically a "random" 0/1 bit string of length $\log_2(3|Q||\Gamma|)$ on left half of tape and move the read/write head back to the separator symbol.

Note that we can assume that the left half of the tape in the original TM remains empty, since for each TM using the left half, we can always construct an equivalent TM which instead uses more space on the right half and leaves the left half of the tape empty.

2nd phase: simulate the original NTM by performing a deterministic run and using the bits on the left half to choose among the non-deterministic transitions of M.

If, in a given configuration, the number of possible transitions in M is smaller than the number read from the tape, M' stops. Although this cannot happen for the original NTM M, the languages accepted by M and M' are the same.

Since M' must always move for each step of M the tape head between the "working space" on the right half of the tape and the non-deterministically written string on the left half, we get: The acceptance of $w \in L$ by M in time $\mathcal{O}(q(n))$ is equivalent to the acceptance of w by M' in time $\mathcal{O}(q(n)^2)$, i. e., polynomially as well.

Hence, without loss of generality, we can assume that an NTM accepting a language L has this two-phase structure. The basic idea of the proof is *not* to represent the *structure* of an NTM by a SAT formula, but instead the *process* of accepting a word from a language $L \in$ NP is encoded in a SAT formula F. For this the tape content after the first phase and the sequence of configurations visited in the second phase is described by F. We follow closely [7].

Theorem: *SAT is NP-complete.*

Proof:

Let $L \in NP$. We have to show $L \leq_P$ SAT. Since we do not know what L looks like, we cannot directly construct a SAT formula from an arbitrary $w \in / \notin L$. We know, however, that there is an NTM $M = (Q, \Sigma, \Gamma, B, q_0, \delta, F)$ which decides $w \in L$ in $p(|w|)$ time, p being a polynomial. We can assume that the NTM is of the "two-phase type" explained above. Let $Q = \{q_0, q_1, \ldots, q_{|Q|-1}\}$ and $\Gamma = \{a_1, \ldots, a_{|\Gamma|}\}$.

Consequently, we have to construct a SAT formula F from M in time polynomial in $|w|$ such that: The NTM M accepts w if and only if F is satisfiable.

First, we change M such that it continues to also work in those states which previously were halt states (without performing any action). Thus, we can assume that *all* computations take exactly $p(|w|)$ steps. This means that $w \in L$ is equivalent to M being at time $t = p(|w|)$ in a (previous) halt state $q \in F$, and the output tape content equals exactly 1.

Second, since the machine performs $p(|w|)$ steps, we know that the head can reach only tape positions within the polynomially-sized range $-p(|w|) \leq j \leq p(|w|)$.

Now we construct a formula, which is true if and only if it represents a valid sequence of configurations of the NTM m, leading to the acceptance of the word w. To be more precise, we start the representation of our computation at the beginning of the second phase (i. e., set the time counter to zero at this time) and describe the content of the tape, which was written in the first phase, by Boolean variables as well. Please note that the number of steps spent in the second phase is strictly less than $p(|w|)$, but it is bounded from above by this number. We can thus continue considering $p(|w|)$ as the actual number of steps.

We want to describe configurations by Boolean variables. For this purpose, we introduce three types of Boolean variables indicating the state, the position of the head and the content of the tape.

- $0 \leq i \leq p(|w|)$, $0 \leq k \leq |Q| - 1$: $Q(i, k) = 1$ iff, at time i, the state is q_k.
- $0 \leq i \leq p(|w|)$, $-p(|w|) \leq j \leq p(|w|)$: $H(i, j) = 1$ iff, at time i, the head is at position j. Position 0 notes the place of the # symbol, at position 1 the input w starts.

- $0 \leq i \leq p(|w|)$, $-p(|w|) \leq j \leq p(|w|)$, $1 \leq l \leq |\Gamma|$: $S(i,j,l) = 1$ iff at time i on position j of the tape, the symbol a_l is stored.

Hence, we have introduced a polynomial number of Boolean variables, their total number is:

$$(p(|w|)+1)|Q| + (p(|w|)+1)(2p(|w|)+1) + (p(|w|)+1)(2p(|w|)+1)|\Gamma|\,.$$

Now we construct the formula F. The following clauses guarantee that the computation and the configurations are valid (1–4), that the initial state (i. e., also the input) is represented correctly (5), and that we end up in an accepting configuration (6). F will be the conjunction of all clauses.

1. At all times i exactly one $Q(i,k) = 1$, i. e., the machine is always in exactly one state. This is guaranteed by the following formula

   $$\bigwedge_i \left[(Q(i,0) \vee Q(i,1) \vee \ldots \vee Q(i,|Q|-1)) \wedge \bigwedge_{k \neq l} (\overline{Q(i,k)} \vee \overline{Q(i,l)}) \right]$$

   The outer conjunction is over all times. The first clause is true, iff at least one $Q(i,k)$ is true, the rest inside the $[\ldots]$ is true if no two $Q(i,k)$ are true at the same time i. Hence the formula is true, if for all i exactly one $Q(i,k)$ is true and all others are false.

   This generates $(p(|w|)+1))\mathcal{O}(|Q|^2) = \mathcal{O}(p(|w|))$ clauses.

2. At all times i, the head is at exactly one position. This means we have one j with $H(i,j) = 1$ and all other $H(i,j)$ are false for a give time i. This creates a set of clauses similar to case 1.

   This generates $(p(|w|)+1))\mathcal{O}(p(|w|)^2) = \mathcal{O}(p(|w|)^3)$ clauses.

3. At each tape position, exactly one letter is stored. This means that at all times i and for all tape positions j exactly one $S(i,j,l) = 1$. Again a formula similar to 1 is created to ensure this.

   This generates $(p(|w|)+1))(2p(|w|)+1))\mathcal{O}(|\Gamma|^2) = \mathcal{O}(p(|w|)^2)$ clauses.

4. In the computation, starting at time 0 (after the random phase has been finished) up to time $p(|w|)$ one configuration follows after the other deterministically, according to the transition function δ.

   Tape fields which are not under the head remain unchanged, which is ensured by satisfying the following formula:

   $$\begin{gathered}\forall 0 \leq i < p(|w|), -p(|w|) \leq j \leq p(|w|), 1 \leq l \leq |\Gamma| : \\ (S(i,j,k) \wedge \overline{H(i,j)}) \Rightarrow S(i+1,j,k)\ .\end{gathered}$$

   Any implication $A \Rightarrow B$ can be written as $\overline{A} \vee B$. Furthermore $\overline{A \wedge B}$ is equivalent to $\overline{A} \vee \overline{B}$, hence we can rewrite the above clause in CNF as

   $$\begin{gathered}\forall 0 \leq i < p(|w|), -p(|w|) \leq j \leq p(|w|), 1 \leq l \leq |\Gamma| : \\ \overline{S(i,j,k)} \vee H(i,j) \vee S(i+1,j,k)\ .\end{gathered}$$

This generates $\mathcal{O}(p(|w|)^2)$ clauses.

Furthermore, the tape-cell under the read/write head, the state and the head positions have to be changed according to the transition function. Please note that in the 2nd phase, the TM works deterministically, hence we can assume $\delta(q_k, a_l) = (q_{c(k,l)}, a_{b(k,l)}, d(k,l))$ were the *functions* $c(k,l)$, $b(k,l)$ and $d(k,l) = +1, 0, +1$ describe how the state, the tape content and the head position are changed. Then we get:

$$\begin{gathered}\forall 0 \leq i < p(|w|),\ -p(|w|) \leq j \leq p(|w|),\ 0 \leq k \leq |Q|-1,\ 1 \leq l \leq |\Gamma| : \\ \overline{H(i,j)} \vee \overline{Q(i,k)} \vee \overline{S(i,j,l)} \vee S(i+1, j, b(k,l)) \\ \overline{H(i,j)} \vee \overline{Q(i,k)} \vee \overline{S(i,j,l)} \vee Q(i+1, c(k,l)) \\ \overline{H(i,j)} \vee \overline{Q(i,k)} \vee \overline{S(i,j,l)} \vee H(i+1, j+d(k,l))\,.\end{gathered}$$

This also generates $\mathcal{O}(p(|w|)^2)$ clauses in all three cases.

5. The initial configuration has to represent the situation after the random phase. This means:

   - The machine is in a specific state (say q_k), the head is in position 0 where the #-symbol is written: $Q(0,k) \wedge H(0,1) \wedge S(0,0,t)$ (we assume $a_t = \#$).
   - On the left half of the tape, the random string is written: for $j < 0 : S(0,j,l_0) \vee S(0,j,l_1) \vee S(0,j,l_2)$ with $a_{l_0} = 0$, $a_{l_1} = 1$, $a_{l_2} = B$ (only 0s, 1s and Bs).
   - To the left of the random strings the tape is empty, i. e., it contains only blanks. In other words, to the left of a blank must be a blank again:

     For $-p(|w|) + 1 \leq j \leq -1$: $S(0,j,l_2) \Rightarrow S(0,j-1,l_2)$ which is equivalent to $\overline{S(0,j,l_2)} \vee S(0,j-1,l_2)$.
   - The input w (w.l.o.g $w \in \{0,1\}^\star$) must be stored initially on the tape as well: $S(0,j,l_0) = \overline{w_j}$ and $S(0,j,l_1) = w_j$ for $1 \leq j \leq |w|$.
   - Finally: Only blanks right to the input: $S(0,j,l_2)$ for $j > |w|$.

   This generates $3 + p(|w|) + p(|w|) - 1 + |w| + p(|w|) - |w| = \mathcal{O}(p(|w|))$ clauses.

6. At the end of the computation, the following requirements have to be fulfilled if the input word is accepted:

   - M must be in a halt state: $\bigvee_{k:q_k \in F} Q(p(|w|), k)$.
   - Under the head there must be written a "1": $\overline{H(p(|w|), j)} \vee S(p(|w|), j, l_1)$.
   - In the field right to the tape head, there must be a symbol not belonging to the input/output alphabet Σ: $\bigwedge_{l:a_l \in \Sigma} \overline{H(p(|w|), j)} \vee \overline{S(p(|w|), j+1, l)}$.

   The second and third case appears for all $-p(|w|) \leq j \leq p(|w|)$, hence the number of clauses is $\mathcal{O}(p(|w|))$.

So we have $\mathcal{O}(p(|w|)^2)$ variables and $\mathcal{O}(p(|w|)^3)$ clauses describing the TM completely, i. e., F can be constructed in a polynomial time. For a given word w the full formula is satisfiable if

and only if, for the NTM, there exists at least one run resulting in output 1, e. g., iff w belongs to L. In this case, the satisfying assignment encodes the accepting run. QED

We thus conclude that any problem in NP can be polynomially mapped onto SAT. In the next section, we will encounter more NP-complete problems. The proof for these problems will be much easier due to the transitivity of polynomial reducibility. Taking any $L \in$ NP, the polynomial reduction $SAT \leq_P L$ is sufficient to show the NP-completeness of language L.

Before we come to this, we want to summarize the situation. We have seen that optimization problems can be regarded as decision problems. We have introduced different classes of decision problems: undecidable, decidable, NP, NP-complete, and P.

As we have said above, P is a subset of NP. It would be nice if one had polynomial algorithms for all problems in NP, in particular also for NP-complete problems. If for one NP-complete problem a polynomial-time decision algorithm could one day be found, then, using polynomial-time reducibility, this algorithm could decide every problem in NP in polynomial time, i. e., P=NP would hold. So far, however, thousands of NP-complete problems have been identified, even more algorithms have been proposed, but nobody has yet found a single polynomial algorithm. Thus, it seems very likely that P≠NP. Note, however, that so far there is no proof for any problem in NP that it is indeed hard and that, consequently, no polynomial algorithm for its solution exists. Therefore, the so-called *P-NP-problem*, whether P≠NP or P=NP, is still unsolved. It is considered one of the most important open problems in modern mathematics, the Clay institute of Mathematics has included it in a list of seven so-called millennium problems.

We can summarize everything we have learned up to now about the different complexity classes, see Fig. 4.4. NP is a subset of the set of decidable problems. The NP-complete problems are a subset of NP. P is a subset of NP. If we assume that P≠NP, then P and NP-complete are disjoint. In this case, there are "intermediate" problems, which are in NP, but neither NP-complete nor in P [8].

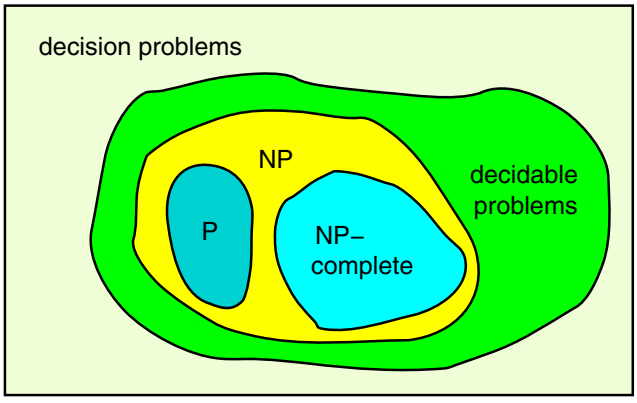


Figure 4.4: Relation between different classes of decision problems (if P≠NP).

4.8 NP-complete problems

Up to now we have seen that there is indeed at least one NP-complete problem. In this section, we will add some other problems. In fact, there is an impressive list of different NP-complete problems in [9], and ever since the publication of this list in 1979, more and more problems have been added.

4.8.1 Proving NP-completeness

You may have some concern that we will present other long and complicated proofs, but that will not be the case. Once a first NP-complete problem is known, we can apply the transitivity of the $\leq_p$ relation to show more easily the NP-completeness of other problems. This gives us the following theorem:

Theorem: *Let L be NP-complete. If $L' \in$NP and $L \leq_P L'$ then L' is NP-complete as well.*

Proof:

Let L be NP-complete, $L' \in$NP and $L \leq_P L'$.

Let $L'' \in$ NP. Since L is NP-complete, we have $L'' \leq_p L$. From the transitivity of $\leq_p$, we get $L'' \leq_p L'$. Hence, we can reduce any problem from NP to L', i. e., L' is NP-complete. QED

This theorem has been used to show for several thousand problems that they are NP-complete. In most cases, the proof is much simpler than Cook's proof for SAT. To start, one has to reduce SAT to the problems L for which we want to prove NP-completeness, since so far SAT is the only NP-complete problem known to us so far.

4.8.2 3-SAT

For practical purposes, 3-SAT is more convenient. Its structure is simpler, since all clauses have three literals. Hence, we first show:

Theorem: *3-SAT is NP-complete.*

Proof:

We show SAT$\leq_p$3-SAT:

Let $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be a Boolean formula in CNF, i. e., every clause C_p contains disjunctions of literals l_{pq}. We are now constructing a 3-SAT formula F^3 which is equivalent to F, hence we want: F satisfiable $\Leftrightarrow$ F^3 satisfiable. This is achieved by replacing each clause C_p by a sequence of 3-clauses in the following way:

- If C_p has three literals, we do nothing.

- If C_p has more than three literals, say $C_p = l_1 \vee l_2 \vee \ldots \vee l_z$ ($z > 3$), we introduce $z-3$ new variables $y_1, y_2, \ldots, y_z$ and replace C_p by the following $z-2$ 3-clauses $(l_1 \vee l_2 \vee y_1) \wedge (\overline{y_1} \vee l_3 \vee y_2) \wedge \ldots (\overline{y_{z-3}} \vee l_{z-1} \vee l_z)$.

  Now assume that $C_p = 1$ (i. e., C_p is true), then at least one $l_p = 1$. Now we choose $y_i = 1$ for all $i \leq p-2$ and $y_i = 0$ for all $i > p-2$. Then all new $z-2$ clauses are true. On the other hand, if the conjunction of the $z-2$ clauses is true, there must be at least one $l_i = 1$. Consequently, if C_p is satisfiable, then all new clauses are satisfiable as well and vice versa.

- Last but not least, C_p may have less than three literals. If $C_p = l$ we replace it by $l \vee l \vee l$ and if $C_p = l_1 \vee l_2$ we replace it by $l_1 \vee l_2 \vee l_2$.

In the end we have a 3-CNF formula F^3 which is (un)satisfiable iff F is (un)satisfiable. The construction of F^3 obviously works in polynomial time. Consequently, SAT $\leq_p$ 3-SAT, and 3-SAT is NP-complete. QED

Note that 2-SAT is *not* NP-complete. Indeed, 2-SAT belongs to P, i. e., there is a polynomial algorithm, which decides for each 2-SAT formula, whether it is satisfiable or not. We will present the algorithm in Sec. 10.1.1.

One can intuitively understand that 2-SAT is not NP-complete from looking again at the proof that 3-SAT is NP-complete. For the case of clauses having $z > 3$ literals, each clause was replaced by a sequence of 3-literal clauses. In many of these clauses, two new literals $\overline{y}_{i-1}$ and y_i where used to "connect" to the preceding and to the following clause, the third literal l_{i+1} was a literal, which has been present in the original clause as well. For a 2-literal clause on the other hand, it is impossible to include the two necessary "connecting" literals and one original literal as well, as needed to represent the original clause. This illustrates why 2-SAT is not NP-complete.

4.8.3 Vertex cover

In the central part of this book, we will work with the *vertex-cover* problem (VC), which has been defined in Sec. 3.1.4. As a reminder: Given a graph $G = (V, E)$, we are looking for a subset $V' \subset V$ of vertices, such that each edge is incident to at least one vertex from V': $\forall \{i, j\} \in E : i \in V' \vee j \in V'$. The minimization problem requires one to search a vertex cover V' of minimum cardinality $|V'|$. The corresponding decision problem poses the question of whether there is a vertex cover of size smaller than or equal to a given threshold K.

Later on in this book, we will study VC numerically as well as analytically with methods from statistical physics. Furthermore, we will analyze the typical, i. e., median, performance of several VC algorithms over a suitably parameterized ensemble of random graphs. We will show that, although VC is hard in the *worst case*, there are some regions in parameter space were VC is *typically* easy, i. e., polynomially solvable, while there are other regions where VC is typically hard. These two regions are separated by a phase transition. Even if obtained for one specific NP-complete problem, these results are very exciting. They allow us to learn

a lot about decision and optimization problems in general. This is the case because VC is equivalent to all other hard problems, i. e., VC is NP-complete.

Now we show that VC is NP-complete. This will be the last section where you are confronted with techniques from theoretical computer science. Once we have proved the NP-completeness of VC, we can concentrate on algorithms for deciding VC, on analytical methods from statistical physics for solving it and on the analysis of the resulting phase diagram.

Theorem: *VC is NP-complete [9].*

Proof:

Obviously, VC is in NP. It is very easy to decide, for a given subset V' of at most K vertices, whether all edges are covered, i. e., whether V' is a vertex cover, by just iterating over all edges.

Hence, it remains to show that 3-SAT is polynomially reducible to VC, i. e., 3-SAT$\leq_p$ VC.

Let $F = C_1 \wedge \ldots \wedge C_m$ be a 3-SAT formula with variables $X = \{x_1, \ldots, x_n\}$ and $C_p = l_p^1 \vee l_p^2 \vee l_p^3$ for all p. We have to create a graph G and a threshold K, such that G has a VC of size lower than or equal to K, iff F is satisfiable. For this purpose, we set:

- $V_1 \equiv \{v_1, \overline{v}_1, \ldots, v_n, \overline{v}_n\}$, ($|V_1| = 2n$) and $E_1 = \{\{v_1, \overline{v}_1\}, \{v_2, \overline{v}_2\}, \ldots, \{v_n, \overline{v}_n\}\}$, i. e., for each variable occurring in F we create a pair of vertices and an edge connecting them.

  To cover the edges in E_1, we have to include at least one vertex per pair in the covering set. In this part of the graph, each cover corresponds to an assignment of the variables. If for variable $x_i = 1$, then v_i should be covered, while if $x_i = 0$ then $\overline{v}_i$ is to be covered. Why this correspondence is chosen, will now become clear.

- For each clause in F we introduce three vertices connected in the form of a triangle: $V_2 \equiv \{a_1^1, a_1^2, a_1^3, a_2^1, a_2^2, a_2^3, \ldots a_m^1, a_m^2, a_m^3\}$ and $E_2 = \{\{a_1^1, a_1^2\}, \{a_1^2, a_1^3\}, \{a_1^3, a_1^1\}, \{a_2^1, a_2^2\}, \{a_2^2, a_2^3\}, \{a_2^3, a_2^1\}, \ldots, \{a_n^1, a_n^2\}, \{a_n^2, a_n^3\}, \{a_n^3, a_n^1\}\}$.

  Per triangle, i. e., per clause, we have to include at least two vertices in a VC. In a cover of minimum size, the *uncovered* vertex corresponds to the literal which is satisfied, as induced by the following edges generated.

- Hence, for each position i in a clause p, vertex a_p^i is connected with the vertex representing the literal f_p^i appearing at that position of the clause: $E_3 \equiv \{\{a_p^i, v_j\} | p = 1, \ldots, m,\ i = 1, 2, 3 \text{ if } l_p^i = x_j\} \cup \{\{a_p^i, \overline{v}_j\} | p = 1, \ldots, m,\ i = 1, 2, 3 \text{ if } l_p^i = \overline{x}_j\}$. Hence, E_3 contains only edges connecting one vertex from V_1 with one vertex from V_2.

- The graph G is the combination of the above introduced vertices and edges: $G = (V, E)$, $V = V_1 \cup V_2$, $E = E_1 \cup E_2 \cup E_3$.

- $K \equiv n + 2m$.

In the following example, we show how the transformation works for a small 3-SAT formula.

Example: 3-SAT$\leq_p$VC

We consider $F = (x_1 \vee \overline{x}_3 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_4)$. F has $n = 4$ variables and $m = 2$ clauses.

The resulting graph $G(V, E)$ is displayed in Fig. 4.5 together with a vertex cover of size $K = n + 2m = 8$. □

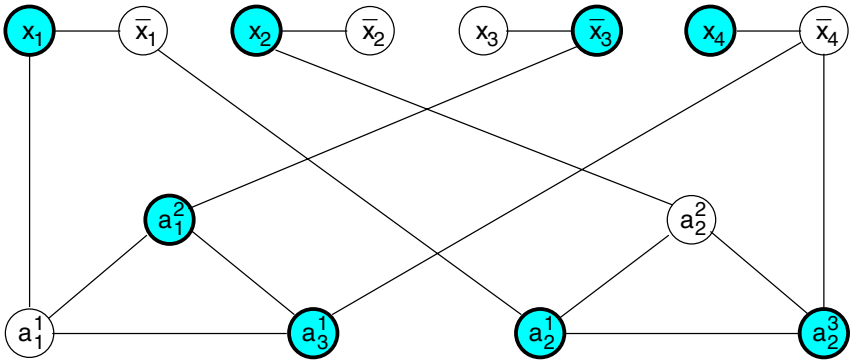


Figure 4.5: VC instance resulting from the 3-SAT instance $F = (x_1 \vee \overline{x}_3 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_4)$. A vertex cover is shown, the covered vertices are indicated by filled circles.

The transformation generates a number of vertices which is $\mathcal{O}(n + m)$, i. e., linear in the sum of the number of clauses and the number of variables of F. Since the number of variables is bounded by three times the number clauses, the construction of the graph is linear in the length of F, i. e., in particular polynomial. It remains to show: F satisfiable $\Leftrightarrow$ there exists a vertex cover V' for G with size $|V'| \leq K$.

($\rightarrow$) Now let F be satisfiable and $\{X_i\}, X_i = 0, 1$ a satisfying assignment. We set $V_1' = \{v_i | X_i = 1\} \cup \{\overline{v}_i | X_i = 0\}$. Obviously $|V_1'| = n$ and all edges in E_1 are covered. For each clause C_p, since it is satisfied by $\{X_i\}$, there is one satisfied literal $l_p^{i(p)}$. We set $V_2' = \{a_p^i | p = 1, \ldots, m;\ i \neq i(p)\}$. We have included 2 vertices per clause in V_2 (by excluding $a_p^{i(p)}$), i. e., 2 vertices per triangle in E_2. Thus, $|V_2'| = 2m$ and all edges of E_2 are covered. Furthermore, since $l_p^{i(p)}$ is satisfied, the vertex corresponding to the literal is in V_1, hence all edges contained in E_3 are covered as well. To summarize $V' = V_1' \cup V_2'$ is a VC of G and $|V'| = n + 2m \leq K$.

($\leftarrow$) Conversely, let be $V' \subset V$ be a VC of G and $|V'| \leq K$. Since a VC must include at least one vertex per edge from E_1 and at least two vertices per triangle from E_2, we know $|V'| \geq n + 2m = K$, hence we have $|V'| = K$, i. e., *exactly one* vertex per pair $x_i, \overline{x_i}$ and

exactly two vertices per triplet a_p^1, a_p^2, a_p^3 is included in V'. Now we set $X_i = 1$ if $x_i \in V'$ and $X_i = 0$ if $x_i \notin V'$. Since each triangle (each corresponding to a clause), has one vertex $a_p^i(p) \notin V'$, we know that the vertex from V_1 connected with it is covered. Hence, the literal corresponding to this vertex is satisfied. Therefore, for each clause, we have a satisfied literal, hence F is satisfied and $\{X_i\}$ is a satisfying assignment. QED

To conclude this chapter, we just mention some other NP-complete problems, which we have already encountered. First, the Hamiltonian cycle Problem (HC) is NP-complete. This can be shown by reduction of 3-SAT (3-SAT$\leq_p$HC). Since we know HC$\leq_p$TSP from page 83, we obtain that TSP is NP-complete as well. Finally, we mention that the Coloring Problem is also NP-complete.

4.9 Worst-case vs. typical-case complexity

In this chapter, we have classified problems according to the scaling of their algorithmic solution time with the input size of the problem instances. To be more precise, this concept is related to a *worst-case scenario*. If there are very, very few but very hard instances, which exist for most input sizes and which need a huge solution time, they will dominate the complexity of the full problem under consideration.

Over recent years, however, computers are becoming more and more influential in many aspects of our daily life. This means, in this context, that numerical solutions for combinatorial optimization and decision problems ranging from scheduling, industrial product design to economical decision-taking, are becoming increasingly important. This is also true for different aspects of scientific research, e. g., computational physics is a quite recent, rapidly growing area. A second example is biology which, over the last about two decades, has collected and is still collecting an incredible amount of data – which has to be exploited numerically.

In the context of such applications, the importance of worst-case results has been questioned. It is, e. g., not clear how frequent the worst-case instances are, and if they are relevant at all for practical questions. The necessity of developing some kind of *typical-case complexity theory* has become clear. One important step in this direction is the study of suitably parametrized random ensembles of input instances. Instead of considering, e. g., the worst-case performance of a vertex-cover algorithm over all graphs of N vertices and M edges, median running times over the random graph ensemble $\mathcal{G}(N, M)$ can be studied. This excludes the strong influence of very rare problem instances, and allows us to make statements valid for almost all graphs (with the notation "almost all" being used in the sense introduced in the random-graph section. The probability that some property holds tends to one for growing graph order $N \to \infty$). This alternative approach has allowed one to find extremely interesting results. One finds, e. g., phase transitions from parameter regions (e. g., classified using M/N for random graphs) of the random ensemble of problem instances where the decision problems are almost always answered with "yes", to regions where the answer is almost surely "no". Far away from these phase transitions, problems seem to become typically easy – a solution is either constructed easily, or contradictions are found efficiently. In these regions, the problems

are almost surely solvable in polynomial time, even if their worst-case running time is still expected to be exponential (unless P=NP). The hardest problems to solve are found close to these phase boundaries. We have already encountered an example for such behavior in Chap. 1 for the random TSP. The study of random input instances is obviously a perfect application area for statistical-physics tools, in particular for those coming from the field of disordered systems. It will therefore be the focus of all following chapters of this book.

On the other hand, one may criticize the fact that random ensembles like $\mathcal{G}(N, M)$ are unrealistic models for real-world applications, where input instances may have, e. g., some specific structure which is far from being completely random. Even if this criticism is obviously sensible, random input instances form, however, a very useful complementary point of view to the classical worst-case scenario. As long as some "real-case complexity theory" is completely missing, there cannot be any doubt about the usefulness of expanding the knowledge about random input instances.

Bibliography

[1] A. M. Turing, Proc. London Math. Soc. **42**, 230; **43** 544 (1936).

[2] C. H. Papadimitriou, *Computational Complexity* (Addison-Wesley, 1994).

[3] M. Presburger, Comptes Rendus du I congrés de Mathèmaticiens des Pays Slaves, 92–101 (1929). [Mojezesz Presburger, born 1904, died probably 1943 in the Warschauer Getto].

[4] M. J. Fischer, M. O. Rabin, Proceedings of the SIAM-AMS Symposium in Applied Mathematics **7**, 27 (1974).

[5] Wikipedia, the free encyclopedia, see
`http://www.wikipedia.org/wiki/Presburger_arithmetic`

[6] S. A. Cook, Ann. ACM Symp. on Theory of Computing, 151 (Assoc. for Comp. Mach., New York 1971).

[7] I. Wegener, *Theoretische Informatik – eine algorithmenorientierte Einführung*, (Teubner, Stuttart, Leipzig 1999).

[8] R. E. Ladner, J. Assoc. Comput. Mach. **22**, 155 (1975).

[9] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*, (W. H. Freeman & Company, San Francisco 1979).

5 Statistical mechanics of the Ising model

In this chapter, some basic aspects of statistical mechanics are discussed. The scope is to provide language and methods which are needed to understand the statistical mechanics approach to combinatorial optimization problems, which will be discussed in the forthcoming chapters. For a general introduction to statistical mechanics, we refer the reader to specialized textbooks [1–3]. We start our discussion with a short presentation of some basic concepts and with the solution of the Weiss model of a ferromagnet, which belongs to the most fundamental but also most easily accessible model in statistical mechanics. In the main part of this chapter, we consider a ferromagnetic Ising model on a random graph, and solve it using the two most important methods in the statistical physics of disordered systems. First, we discus the replica approach, then the Bethe–Peierls approach, which can be seen as a special case of the cavity approach. These models will already demonstrate the close connection between the structure of the underlying random graph and the thermodynamic behavior of the model, which gives one of the major motivations for using statistical-physics tools in analyzing combinatorial problems which, *a priori*, are not based on physical systems.

5.1 Phase transitions

Many physical systems show a drastic change in their properties when some external parameters are slightly modified – they change their macroscopic state and undergo a *phase transition*. Well-known examples are the following.

Liquid–gas transition

The best-known example is water, which is boiling at $100\ ^\circ\mathrm{C}$ (at normal pressure). Above this temperature, water forms a gas (vapor), below, it is a liquid, see Fig. 5.1 for a schematic picture. Exactly at the transition point, there is a coexistence of both liquid water and vapor – the relative fractions depending, e. g., on the total amount of water and the occupied volume. When plotting the density as a function of the temperature, one sees a jump at the transition temperature. In physical terms, we speak of a *first-order phase transition*.

Melting transition

The situation is very similar at 0 °C, where water becomes a solid – ice. The coexistence of water and ice at the phase transition is known to everybody from ice-cubes in soda.

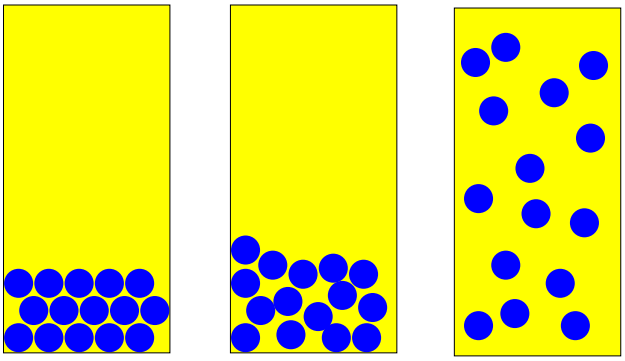


Figure 5.1: Schematic display of the three states of water, frozen at temperatures below the melting temperature T_m (left), liquid at intermediate temperatures (middle), and gas for temperatures $T > T_g$.

Ferromagnetic transition

If a piece of iron is brought close to a permanent magnet, it is strongly attracted. Materials showing a similar behavior are called *ferromagnets*. If, however, the temperature of iron is increased above $T_c = 770$ °C, the attraction is lost, and iron becomes *paramagnetic*. The attraction goes down continuously if we approach T_c from below, and there is no coexistence of the two phases. This leads to a continuous behavior of the magnetization as a function of the temperature around the transition, see also Fig. 5.4 below. One says that the transition is of *second order*.

The transition temperatures depends strongly on the material, but the behavior close to this transition, does not depend on the specific material under consideration – the behavior is *universal*. This is visible by measuring *critical exponents*, which describe the algebraic behavior of measurable quantities as a function of the distance to the critical point T_c. Usually critical exponents do not depend on details of the material, i. e., they are universal. This concept is very important in statistical physics because it allows one to neglect many specific features of a material and to give a very general description of a phase transition for a broad class of substances.

Denaturation transition of DNA

Phase transitions are not restricted to traditional physical systems, their analysis becomes more and more important also in interdisciplinary fields. The first example is taken from biology or biophysics.

Under physiological conditions, DNA (Deoxyribonucleic acid) has the structure of a double helix as discovered by Watson and Crick in 1953. Two long anti-parallel chains are connected by hydrogen bonds. This double helical structure is important for the stable conservation of the genetic code which is stored in the DNA. Mutations or other "damages" to a single strand of DNA can be repaired using the information of the second strand.

If the temperature is increased, if the ionic concentration is changed, or if mechanical torque is imposed on the helix, the molecule may undergo a violent change in structure: The hydrogen bonds may break and double-stranded DNA (dsDNA) undergoes a transition to two single strands (ssDNA).

A local opening of dsDNA by enzymes, which is important for the reading of the genetic code or the duplication of DNA is not caused by a phase transition, because the global structure remains unchanged.

Denaturation also plays a role in the spatial folding structure of proteins (secondary and tertiary structure). Misfolded proteins unfortunately became recently known to be responsible for the mad-cow and Creutzfeldt-Jacob diseases.

Transitions in combinatorial systems

Mathematical systems also undergo phase transitions. We have already considered, in Chap. 3, two examples in connection with random graphs, namely the appearance of the giant component and formation of the q-core of a random graph. There the changed parameter was the average vertex degree of the graphs, and it plays a similar role to temperature or ionic concentration in previous examples.

In this book we will see and study other examples of phase transitions in combinatorial problems. We will also see how the ideas developed to analyze physical systems like ferromagnets can be used to describe combinatorial problems.

Microscopic versus macroscopic description

What is common to all these examples?

In all cases, the systems are microscopically unchanged (or only slightly changed). The iron crystal, e. g., is not changed at the ferromagnetic transition. On the other hand, the macroscopic properties, which may be of physical, chemical, biological or purely mathematical nature, are completely different in the two different phases.

In all cases, this change appears when the control parameters are modified and cross some isolated critical value. In most cases, the control parameter is given by the temperature, but it can be much more general as we have seen above. It is the objective of statistical physics, to understand why the macroscopic properties of a system may undergo such violent changes if the control parameters are changed only slightly.

5.2 Some general notes on statistical mechanics

5.2.1 The probability distribution for microscopic configurations

The aim of statistical physics is to derive the macroscopic behavior of a systems starting from a description of its microscopic constituents, together with their pairwise interactions. In mechanics, one would try to describe the system by the trajectories of all particles – which is impossible if we speak about 10^{23} constituents.

Intuitively it is, however, clear that the macroscopic behavior cannot depend on the specific trajectory taken by a single particle, but only on the statistical properties of all trajectories.

The main idea in statistical mechanics is that every microscopic configuration $\mathcal{C}$ (e. g., the particle positions or spin orientations) is assigned a probability $p(\mathcal{C})$ which depends on its *energy*, as given by the *Hamiltonian* $H(\mathcal{C})$. This means that, taking a momentary "photograph" of the microscopic configuration of a system, it is found with probability $p(\mathcal{C})$ to be in configuration $\mathcal{C}$. Here we consider the *canonical ensemble*, i. e., a system which is coupled thermally to a heat bath held at temperature T. This means that the probability $p(\mathcal{C})$ depends also on the temperature and is given by the Gibbs distribution

$$p(\mathcal{C}) = \frac{1}{Z} \exp\left\{-\frac{1}{T} H(\mathcal{C})\right\} \tag{5.1}$$

where the normalization factor

$$Z = \sum_{\mathcal{C}} \exp\left\{-\frac{1}{T} H(\mathcal{C})\right\} \tag{5.2}$$

is called the *(canonical) partition function*. In the last equation we have used a discrete summation over all configurations, which has to be replaced by an integration in the case of continuous degrees of freedom. In the following, we are, however, much more interested in combinatorial systems which are characterized by discrete variables and thus a discrete configuration space.

The results of (5.1) for two extreme cases are obvious:

- Temperature $T = \infty$: The probability $p(\mathcal{C})$ becomes independent of the energy $H(\mathcal{C})$, each configuration $\mathcal{C}$ is equiprobable. The system is in a totally disordered phase, like a hot gas.
- Temperature $T = 0$: The probability $p(\mathcal{C})$ is completely concentrated in the global minima of $H(\mathcal{C})$, i. e., the *ground states*. The system is frozen into these states. Usually, ground states exhibit a higher degree of order, as for a perfect crystal, a magnetized state or for the double-stranded DNA.

In between these two limits, the probability $p(\mathcal{C})$ depends on the energy, with $p(\mathcal{C})$ being higher for lower energies $H(\mathcal{C})$. The relative differences of the probability between neighboring energies increase for lower temperatures, explaining the tendency to higher order.

5.2.2 Statistical meaning of the partition function

The average energy is defined by

$$\langle E\rangle_T = \sum_{\mathcal{C}} p(\mathcal{C})H(\mathcal{C}) \tag{5.3}$$

but can also be calculated from the partition function via

$$\langle E\rangle_T = -\frac{d}{d\beta}\ln Z\ , \tag{5.4}$$

with $\beta = 1/T$, which can easily be verified from Eq. (5.2), i. e., as a derivative of the *free energy*

$$F = -T\ln Z\ . \tag{5.5}$$

The second derivative of F describes the energy fluctuations:

$$\langle(\Delta E)^2\rangle_T = \langle E^2\rangle_T - \langle E\rangle_T^2 = -\frac{d^2}{d\beta^2}\ln Z\ . \tag{5.6}$$

This suggests that the partition function can be seen as a generating function of energies. In fact we can rewrite

$$\begin{aligned} Z &= \sum_{\mathcal{C}} \exp\{-\beta H(\mathcal{C})\} \\ &= \sum_{E} \mathcal{N}(E)\exp\{-\beta E\} \end{aligned} \tag{5.7}$$

where $\mathcal{N}(E)$ is the number of configurations $\mathcal{C}$ having exactly energy $E = H(\mathcal{C})$. The logarithm of this number gives the *microcanonical entropy* $S(E) = \ln\mathcal{N}(E)$ of configurations of energy E. Since the energy fluctuates in a canonical ensemble, it is reasonable also to define the average entropy $\langle S\rangle_T$ which is related to the free energy and the average energy by

$$F = \langle E\rangle_T - T\langle S\rangle_T\ . \tag{5.8}$$

We refer the reader to textbooks [1–3] where it is shown that this definition of the entropy and the microcanonical entropy are equivalent. In the canonical ensemble, the systems tends to minimize its free energy. Thermodynamically, the entropy is minimal for minimal energy. This leads to a competition between the energy and entropy in Eq. (5.8): The minimization of F asks for a minimization of the energy – which decreases the entropy – and also for a maximization of the entropy (due to the minus sign) – which would increase the energy. The balance of these which tend to the opposite gives exactly the thermodynamic equilibrium. Looking to Eq. (5.8), it becomes clear that energetic contributions become more influent for low temperature T, whereas entropic contributions dominate for high T. This again explains the tendency to higher order at low temperatures.

5.2.3 Thermodynamic limit

In statistical mechanics, as the name suggests, we are interested in the behavior of large systems. The reason is that a small sample of matter usually contains a large number of atoms, e. g., of the order of 10^{23}. Technically, we handle this by calculating properties of systems of a finite number of N particles, and then we perform the *thermodynamic limit* $N \to \infty$. This has to be done in such a way, that *intensive* quantities, like the energy per particle, remain finite, although *extensive* quantities, like the total energy, will diverge like $\mathcal{O}(N)$. On the other hand, this limit allows us in the subsequent chapters to ignore the contributions which exhibit a sub-dominant contribution to any calculated quantity, e. g., a $(\ln N)/N$ behavior for an intensive quantity.

5.3 The Curie–Weiss model of a ferromagnet

In classical statistical mechanics, ferromagnetism is frequently described in terms of the *Ising model* [4]. The local magnetic moments are represented by *Ising spins* $\sigma_i = \pm 1$, with $i = 1, \ldots, N$. A ferromagnetic interaction of two Ising spins means that a parallel orientation is favored, i. e., the model can be described by the Hamiltonian

$$H(\sigma_1, \ldots, \sigma_N) = -\sum_{\langle i,j \rangle} \sigma_i \sigma_j \tag{5.9}$$

where the sum $\langle i, j \rangle$ is restricted for simplicity over all pairs of nearest neighbors of the lattice under consideration. In the two ground states, $\{\sigma_i \equiv 1,\ i = 1, \ldots, N\}$ and $\{\sigma_i \equiv -1,\ i = 1, \ldots, N\}$, all interactions are simultaneously *satisfied*, i. e., for all interactions the resulting energy contribution is minimal. In one dimension, it is known that the model has no phase transition at finite temperature [4]. In two dimensions, according to the famous solution of Onsager given in 1944 [5], there exists a phase transition at a finite temperature. This is true also for higher dimensions, where the existence of a phase transition is rigorously established. Nevertheless, the three-dimensional Ising model is still analytically unsolved and the calculation of its free energy belongs to the big open questions in statistical physics. Hence, many results for three dimensions were obtained using numerous numerical investigations.

In order to qualitatively describe the behavior at the ferromagnetic transition, an infinite-dimensional toy model will be considered in the following. Even if this model is not interesting from the point of view of combinatorial optimization – the ground states are known – the analysis will provide us with many techniques which are important for the following sections. In the toy model, not only do nearest-neighbor spins interact, but each spin is connected to all other spins, i. e., the Hamiltonian is modified to

$$H(\sigma_1, \ldots, \sigma_N) = -\frac{1}{N} \sum_{i<j} \sigma_i \sigma_j \ . \tag{5.10}$$

In this sense, the model is defined on the complete graph K_N of N vertices and $N(N-1)/2$ edges. The multiplier $1/N$ in the definition of H ensures that energies are extensive, i. e., of $\mathcal{O}(N)$.

As mentioned before, the main task of equilibrium statistical mechanics consists of calculating the partition function of the model, from which the free energy as a thermodynamic potential can be read off. Before doing this, we slightly rewrite the Hamiltonian,

$$\begin{aligned} H(\sigma_1, \ldots, \sigma_N) &= -\frac{1}{N} \sum_{i<j} \sigma_i \sigma_j \\ &= -\frac{1}{2N} \left(\sum_i \sigma_i \right)^2 + \frac{1}{2N} \sum_i \sigma_i^2 \\ &= -\frac{N}{2} m^2 + \mathcal{O}(1) \,. \end{aligned} \tag{5.11}$$

In the last step, we have introduced the *magnetization*

$$m = \frac{1}{N} \sum_i \sigma_i \tag{5.12}$$

which takes values in $\{-1, -1+2/N, -1+4/N, \ldots, +1\}$. This quantity is also called an *order parameter*, because it is typically zero in the *paramagnetic* (i. e., random) state, and it is typically different from zero in the *ferromagnetic* state. In the ground states, it takes its extreme values ± 1. Using this, and neglecting non-extensive terms in H, we can rewrite the partition function

$$\begin{aligned} Z &= \sum_{\{\sigma_i\}} e^{-\beta H(\sigma_1, \ldots, \sigma_N)} \\ &= \sum_{\{\sigma_i\}} \sum_m \delta_{m, \sum_i \sigma_i / N} \, e^{\beta N m^2 / 2} \end{aligned} \tag{5.13}$$

The dependency on the local spins σ_i is now completely reduced to the Kronecker symbol fixing the magnetization. We will use this type of rewriting several times in subsequent chapters. Interchanging the sum over spin configurations and all possible values of the magnetization m, we have to calculate the number of configurations $\{\sigma_i\}$ leading to a given magnetization m. This is very simple since each configuration with $N_+ = N(1+m)/2$ spins of value $+1$ and $N - N_+$ spins of value -1 leads to magnetization m, i. e., the total number of such a configuration equals the number of possible choices of the N_+ positive spins among all N spins. We thus have

$$\mathcal{N}(m) = \sum_{\{\sigma_i\}} \delta_{m, \sum_i \sigma_i / N} = \binom{N}{N \frac{1+m}{2}} = \frac{N!}{\left(N \frac{1+m}{2}\right)! \left(N \frac{1-m}{2}\right)!} \,. \tag{5.14}$$

Using Stirling's formula $\ln N! \simeq N \ln N - N$, we obtain in general

$$\begin{aligned}\binom{N}{\gamma N} &= \frac{N!}{(\gamma N)!((1-\gamma))N!} \\ &\simeq \exp\{N\ln N - N - (\gamma N)\ln(\gamma N) + \gamma N \\ &\quad -((1-\gamma)N)\ln((1-\gamma)N) + (1-\gamma)N\} \\ &= \exp\{-N\gamma\ln\gamma - N(1-\gamma)\ln(1-\gamma)\}\ . \end{aligned} \tag{5.15}$$

Hence, $\mathcal{N}(m)$ evaluates to

$$\mathcal{N}(m) = \exp\left\{-N\frac{1+m}{2}\ln\frac{1+m}{2} - N\frac{1-m}{2}\ln\frac{1-m}{2}\right\}\ . \tag{5.16}$$

This has to be plugged into Eq. (5.13). The sum over all m-values can be interpreted as a Riemann sum (with a missing factor $\Delta m = 2/N$ which gives negligible logarithmic corrections to the free energy), and it can be replaced by an integration for $N \gg 1$. We thus find

$$\begin{aligned} Z &= \int_{-1}^{1} dm\ \exp\left\{-N\left[\frac{1+m}{2}\ln\frac{1+m}{2} + \frac{1-m}{2}\ln\frac{1-m}{2} - \frac{\beta}{2}m^2\right]\right\} \\ &\equiv \int_{-1}^{1} dm\ \exp\{-\beta N f(m)\}\ . \end{aligned} \tag{5.17}$$

In this expression, the sum over all microscopic configurations is carried out, only a single integration over the order parameter is left.

How does one solve this integral? Obviously, this cannot be done completely generally, but we are only interested in the exponentially dominating contribution, or, equivalently, in the extensive contributions to the free energy. These can be evaluated using the *saddle-point approximation*, the second key method used in the forthcoming chapters.

$$Z = \int_{-1}^{1} dm \exp\{-\beta N f(m)\} = \exp\{-\beta N \min_m f(m) + \mathcal{O}(\ln N)\} \tag{5.18}$$

which will be justified below. This means, to leading order and for large N, the full integral can be replaced by the maximum of the integrand. This can be seen in the following way. The largest contributions to Z are expected to result from the vicinity of the maximum of the integrand, i. e., we can expand $f(m)$ around its absolute minimum (argument m^*). To second order we have

$$f(m) = f(m^*) + \frac{1}{2}f''(m^*)(m-m^*)^2 + \mathcal{O}((m-m^*)^3)\ . \tag{5.19}$$

Keeping only the first two terms in I, the integral becomes Gaussian. We assume for a moment that we can extend the limits of the integral to $\pm\infty$, then it can be evaluated using

$\int \exp(-\frac{x^2}{2\sigma})\,dx = \sqrt{2\pi\sigma}$:

$$\begin{aligned} Z &\simeq \int dm \exp\{-\beta N(f(m^*) + \frac{1}{2}f''(m^*)(m-m^*)^2)\} \\ &= e^{-\beta N f(m^*)} \sqrt{\frac{2\pi}{\beta N f''(m^*)}}\,. \end{aligned} \tag{5.20}$$

Including the last factor into the exponential, it becomes logarithmic in N, even with the extended integration range. Hence, it is also subdominant for $m \in [-1,1]$, which confirms Eq. (5.18). Internal extrema of $f(m)$ are characterized by a vanishing derivative with respect to m. Taking the specific form of $f(m)$ given in Eq. (5.17), we have to solve the *saddle-point equation*

$$0 = \beta f'(m^*) = \frac{1}{2}\ln\frac{1+m^*}{2} + \frac{1}{2} - \frac{1}{2}\ln\frac{1-m^*}{2} - \frac{1}{2} - \beta m^*\,, \tag{5.21}$$

i. e.,

$$\beta m^* = \frac{1}{2}\ln\frac{1+m^*}{1-m^*} = \ln\sqrt{\frac{1+m^*}{1-m^*}}. \tag{5.22}$$

By exponentiating this equation, we obtain $\exp(\beta m^*)$ and $\exp(-\beta m^*)$, from which we can calculate $\tanh(\beta m^*) = (\exp(\beta m^*) - \exp(-\beta m^*))/(\exp(\beta m^*) + \exp(-\beta m^*))$, resulting in the usual form of the equation

$$m^* = \tanh(\beta m^*)\,. \tag{5.23}$$

The solution(s) of this saddle-point equation are best discussed graphically, see Fig. 5.2. For $T > 1$, or $\beta < 1$, the slope of the hyperbolic tangent in $m = 0$ is less than one, and there is only one solution to Eq. (5.23): $m^* = 0$. The system is globally unmagnetized and in its paramagnetic phase.

The existence of this solution is clear from the very beginning, because it reflects the *spin-flip symmetry* of the Hamiltonian $H(\sigma_1, \ldots, \sigma_N)$. It remains invariant if all spins simultaneously change their signs. Since m changes its sign under this transformation, the free-energy function $f(m)$ has to be even in m, and $m = 0$ has to be a local extremum. If it is, in addition, the global minimum and thus, according to the saddle-point approximation, thermodynamically relevant, this symmetry is said to be unbroken also at the level of the order parameter.

If the temperature is below one, i. e., for $\beta > 1$, we find three solutions. The first one is the spin-flip symmetric one, $m = 0$. The other two are non-zero, but differ only in their sign. This is again a result of the symmetry. Applying the symmetry transformation to one solution must lead to another, equivalent one. In order to be able to select the thermodynamically stable, i. e., global minima of $f(m)$, we plot this function in Fig. 5.3. We find that, in fact, there are two equal minima in the two non-zero magnetizations, whereas $m = 0$ becomes a local maximum. The non-zero magnetizations are thus the physical solutions of the saddle-point equation. The system shows a global magnetization, i. e., a ferromagnetic behavior. A single of the two

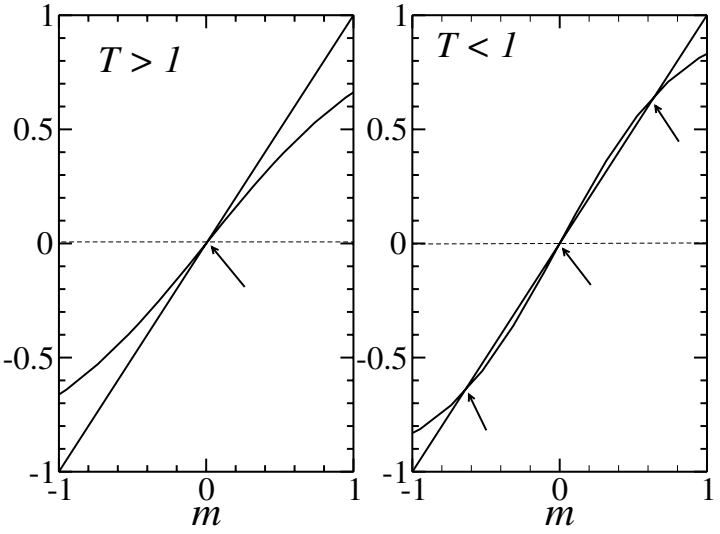


Figure 5.2: Graphical solution of the saddle-point equation for different temperatures. In the high-temperature phase, there is only one solution, in the low-temperature phase, there are three.

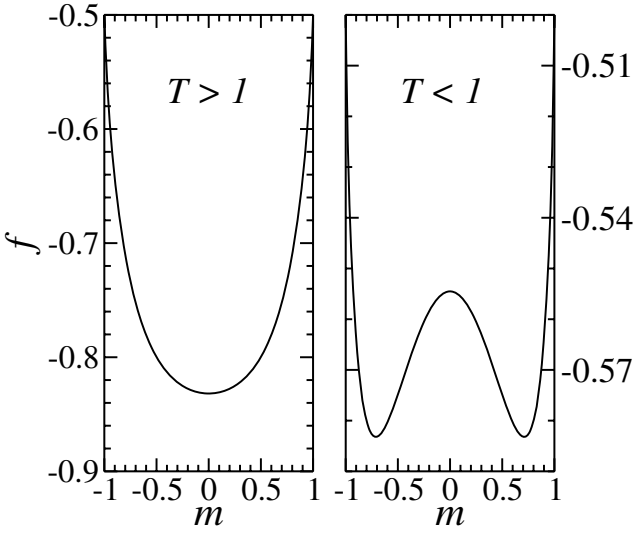


Figure 5.3: Free-energy density for $T > 1$ and $T < 1$. In the first case, there is only one global minimum in $m = 0$, and the system is paramagnetic. In the second case, there are two equal minima of non-zero magnetization which characterize the two ferromagnetic states.

equivalent solutions does not show the spin-flip symmetry of the system. We therefore say that the spin-flip *symmetry is spontaneously broken*.

The paramagnetic and the ferromagnetic phases are thus separated by the *critical temperature* $T_c = 1$. In the definition of the magnetization, we have denoted m as an *order parameter*. The full significance of this name becomes clear if we look at Fig. 5.4. The value of m equals zero in the high temperature phase, and is different from zero in the low-temperature phase. In a more general context, an order parameter is a global observable which takes different values in different phases. The identification of relevant order parameters is of fundamental importance in the description of phase transitions and, in our case, it came out quite naturally from the calculation. As we will see in the next section, order parameters can be much more involved.

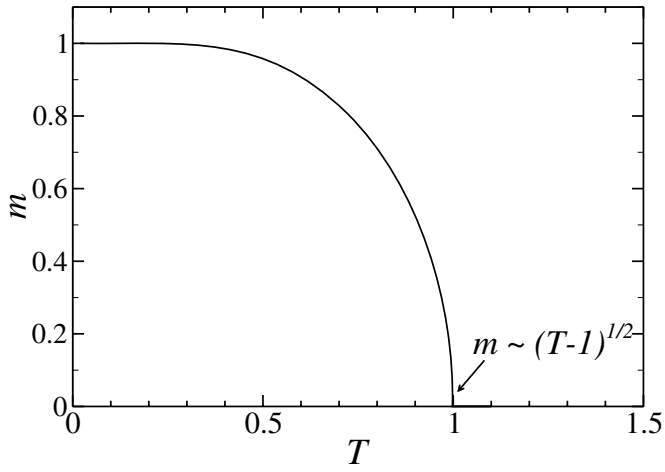


Figure 5.4: Non-negative branch of the thermodynamic solution of the magnetization.

As a last point we are going to discuss the critical behavior, i. e., the growth of the magnetization inside the ferromagnetic phase, but close to the critical temperature. We write $T = 1 - \varepsilon$ with $0 < \varepsilon \ll 1$ which, to leading order, is equivalent to $\beta = 1 + \varepsilon$. Also the magnetization is small, so we can expand the hyperbolic tangent in Eq. (5.23),

$$m = (1 + \varepsilon)m + \frac{1}{3}(1 + \varepsilon)^3 m^3 + \dots \tag{5.24}$$

To leading order we thus find

$$m = \sqrt{3(T - 1)}\,, \tag{5.25}$$

i. e., we find again an algebraic critical behavior. In this case, however, the critical exponent equals 1/2, and the magnetization starts to grow with an infinite slope below $T_c = 1$.

5.4 The Ising model on a random graph

5.4.1 The model

In the last section, we discussed the equilibrium properties of the Ising model on a complete graph. We have seen that, for high temperatures, the system behaves paramagnetically. At a critical temperature we found a phase transition to a ferromagnetic phase.

In this section, we are going to define the Ising model over a more complicated structure, namely a random graph $G = (V, E)$ drawn from the ensemble $\mathcal{G}(N, c/N)$ of graphs of average vertex degree c. We assume that the vertices are $V = \{1, 2, \ldots, N\}$, and edges are drawn independently with probability c/N between any pairs of vertices. The edge set can be uniquely characterized by the symmetric *adjacency matrix* J with entries

$$J_{ij} = \begin{cases} 1 & \text{if} \quad \{i,j\} \in E \\ 0 & \text{if} \quad \{i,j\} \notin E\,. \end{cases} \tag{5.26}$$

The probability density for a given symmetric matrix with vanishing diagonal elements is given by

$$\mathcal{P}(J) = \prod_{i<j} \left[\left(1 - \frac{c}{N}\right) \delta(J_{ij}) + \frac{c}{N} \delta(J_{ij} - 1) \right] . \tag{5.27}$$

The factorized structure of this distribution reflects the independence of different edges.

Given the adjacency matrix J, we can define an Ising model over G by introducing a ferromagnetic interaction for each edge. Denoting the Ising spins again by $\sigma_i = \pm 1, i = 1, \ldots, N$, we write the Hamiltonian

$$H_J(\sigma_1, \ldots, \sigma_N) = -\sum_{i<j} J_{ij} \sigma_i \sigma_j \tag{5.28}$$

which depends on the microscopic configuration $\mathcal{C} = \{\sigma_i\}$ as well as on the so-called *quenched disorder* J. Quenched means frozen in, because the disorder is fixed for one sample, and it is not subject to thermal fluctuations. The main aim of this section is to understand how we can handle systems with such disorder. In addition, the model is defined on a graph type which we studied before, and which we will need later on as well. We will explain some of the statements made there using a statistical-mechanics approach, i. e., for the first time we use statistical physics to analyze a combinatorial problem. This will be done using two different methods, namely the replica and the cavity approach [6].

5.4.2 Some expectations

Let us first formulate some expectations on the behavior of the Ising model which we gain from our knowledge about the graph structure. We will mainly concentrate on ground-state properties, i. e., on zero temperature.

We have already learned in Sec. 3.3 that random graphs undergo a percolation transition at average vertex degree $c = 1$. Below this threshold, almost all vertices belong to small connected components of G, where small means that they connect $\mathcal{O}(N^0)$ vertices. In this case, the ground states of the Hamiltonian (5.28) are trivial for each component. Either all spins are $+1$, or all spins are -1. The spins belonging to different connected components are, however, independent. One component can be fixed to spin value $+1$, another to -1 without violating any (ferromagnetic) bond. Consequently there are $2^{N_{cc}}$ ground states, where N_{cc} denotes the number of distinct connected components of G. If a component has finite size, there is also an energy barrier of only finite size separating the two different ground states of this component. This is not enough to introduce a breaking of the spin-flip symmetry, the barrier can be overcome at arbitrarily small temperature. We therefore conclude that almost any two ground states on a random graph with $c < 1$ are connected by a path crossing only finite energy barriers, and the system has to be paramagnetic down to zero temperature.

If, on the other hand, we have a random graph G with $c > 1$, there exists a giant component which connects a finite fraction of all N vertices. This giant component may introduce ferromagnetic behavior at low temperatures. A conjecture which we will confirm in the calculations presented below. On the other hand, there is still an extensively large number of spins belonging to small connected components of G and these will stay paramagnetic. So there is a simple distinction which allows us to extract the size of the giant component from the analysis of the Ising model at $T = 0$. Ferromagnetically ordered spins belong to the giant component, paramagnetically behaving spins to small subgraphs.

5.4.3 The replica approach

Disorder dependence and self-averaging

Let us now perform the calculation of the partition function in detail. The strategy is similar to the last section where we have analyzed the Weiss model of a ferromagnet. An order parameter will be introduced which allows one to perform the sum over all microscopic configurations. The thermodynamic average of this order parameter will again be evaluated using the saddle-point method. The approach is, however, complicated by the presence of quenched disorder, i. e., by the random structure of the underlying graph $G \in \mathcal{G}(N, c/N)$. The problem results from the fact, that the partition function

$$Z(J) = \sum_{\{\sigma_i\}} \exp\left\{\beta \sum_{i<j} J_{ij}\sigma_i\sigma_j\right\} \tag{5.29}$$

explicitly depends on the adjacency matrix J, i. e., on $\binom{N}{2}$ random numbers.

We profit, however, from an amazing property called *self-averaging*. In the thermodynamic limit, intensive global observables do not depend on the specific choice of the disorder configuration, but only on the statistical properties of the disorder distribution. Thus, e. g., the magnetization of a system is self-averaging, but so is the energy density or the free-energy

density. Technically speaking, the free-energy density approaches its disorder average in the thermodynamic limit, i. e., we can get rid of the disorder by calculating

$$-\beta f = \lim_{N\to\infty} \frac{1}{N} \overline{\ln Z(J)} \tag{5.30}$$

with the over-bar denoting the average over (5.27). This means that we can *first* calculate the disorder average for a finite system, and then, *second*, we can perform the thermodynamic limit.

Note that self-averaging is restricted to *intensive* quantities. It is, in particular, not valid for the partition function itself. So it is not correct to simply calculate the so-called *annealed average* $\ln \overline{Z(J)}$. This would be technically much simpler, but the resulting expression for the free energy would be given by rare graphs which do not describe the typical structure of a $\mathcal{G}(N, c/N)$ graph. In particular, $\overline{\ln Z(J)} \leq \ln \overline{Z(J)}$, i. e., the annealed free energy is overestimated.

The replica trick

The resulting technical problem consists in averaging the logarithm of a sum over all spin configurations. Nobody knows how to achieve this directly. So the calculations are done using the so-called *replica trick* which is a non-rigorous, but well-tested method. It is based on the simple identity

$$\overline{\ln Z} = \lim_{n\to 0} \frac{\overline{Z^n} - 1}{n} \,. \tag{5.31}$$

The idea of the trick is to calculate the right-hand side first for positive *integer* values of n, which is technically relatively easy, and then to analytically continue the results, i. e., to apply the results for continuous non-integer values of n, to let n go to zero. The mathematical problem is that, in general, the integer-n results do not have a unique continuation, and one needs to plug in some heuristic ansatz on the mathematical structure of the order parameters. First attempts were based on the assumption of replica symmetry (cf. below) [7, 8]. A complete and internally consistent ansatz scheme – called replica symmetry breaking – was introduced by G. Parisi in 1979 [9]. The complete proof of the mathematical exactness of this scheme is missing even after a quarter of a century. For so-called infinite-connectivity models, which are defined on the complete graph K_N, the resulting free energy was recently shown to be exact [10]. For so-called finite-connectivity models, including those defined over random graphs, the situation is less satisfactory. Even if the first investigations on these models started a long time ago, see e. g. [11–16] for early approaches, the full Parisi scheme has not yet been realized in these models. Only the first step of replica symmetry breaking is well-understood [17, 18], and it is known to provide a lower bound to the correct free energy [19]. The replica trick was, however, tested successfully in a huge number of models, and there is no real doubt that it gives correct results for the class of so-called mean-field models which include random graphs.

Let us start the calculation for positive integer n. The nth power of the partition function can be written as the n-fold product of $Z(J)$ with itself. We interpret every factor as an independent copy – i. e., a *replica* – of the original system including the same disorder J:

$$\begin{aligned} Z(J)^n &= Z(J)\cdot Z(J)\cdots Z(J) \\ &= \left(\sum_{\{\sigma_i^1\}} e^{-\beta H(\{\sigma_i^1\})}\right)\cdots\left(\sum_{\{\sigma_i^n\}} e^{-\beta H(\{\sigma_i^n\})}\right) \\ &= \sum_{\{\sigma_i^a\}_{a=1,\ldots,n}} \exp\left\{-\beta\sum_{a=1}^{n} H(\{\sigma_i^a\})\right\} \\ &= \sum_{\{\sigma_i^a\}} \exp\left\{\beta\sum_{i<j} J_{ij}\sum_{a=1}^{n}\sigma_i^a\sigma_j^a\right\}. \end{aligned} \tag{5.32}$$

In the last line, and also in the following expressions, the sum over $\{\sigma_i^a\}$ means the sum over all 2^{nN} replicated spin configurations. The average of this expression over the random graph distribution can be easily performed. The average over a sum gives the sum over averages, and each term in the sum factorizes in $i<j$, as well as the disorder distribution (5.27) does. We find

$$\begin{aligned} \overline{Z(J)^n} &= \int\prod_{i<j} J_{ij}\mathcal{P}(J)Z(J)^n \\ &= \sum_{\{\sigma_i^a\}}\prod_{i<j}\left(1-\frac{c}{N}+\frac{c}{N}e^{\beta\sum_a\sigma_i^a\sigma_j^a}\right) \\ &= \sum_{\{\sigma_i^a\}}\prod_{i<j}\exp\left\{-\frac{c}{N}+\frac{c}{N}e^{\beta\vec{\sigma}_i\cdot\vec{\sigma}_j}+\mathcal{O}(N^{-2})\right\} \\ &= \sum_{\{\sigma_i^a\}}\exp\left\{-\frac{cN}{2}+\frac{c}{2N}\sum_{i,j}e^{\beta\vec{\sigma}_i\cdot\vec{\sigma}_j}+\mathcal{O}(N^0)\right\}. \end{aligned} \tag{5.33}$$

Here we have used that the leading terms of the Taylor expansion of the third line give the second line. Higher-order terms have to be compensated by a correction term which, however, is subextensive and thus does not contribute to the free-energy density. Also the diagonal term $i=j$ which is additionally introduced in the last line, contributes only to the $\mathcal{O}(N^0)$-corrections. We have introduced the *replicated spins* $\vec{\sigma}_i=(\sigma_i^1,\ldots,\sigma_i^n)$ having n components ±1. Comparing Eqs (5.32) and (5.33) we see that the original system, with disorder but without interactions between different replicas, is replaced by an averaged, i. e., a homogeneous system with interacting replicas, the effective Hamiltonian is therefore no longer a simple sum over replicas.

Order parameters and saddle-point equations

At this stage, we would like to introduce an order parameter. Some intuition can be gained from the Weiss model considered in the last section. The order parameter there was given by the magnetization, or, equivalently, by the fraction of sites with spin $+1$. In the present case we do not have single spins on each site, but replicated ones. The corresponding generalization of the order parameter would be to introduce the fractions of sites i having replicated spin $\vec{s} \in \{\pm 1\}^n$, as suggested in [17],

$$c(\vec{s}) = \frac{1}{N} \sum_i \delta_{\vec{s},\vec{\sigma}_i} \tag{5.34}$$

where $\delta_{.,.}$ is the n-dimensional Kronecker symbol, which we will denote below also as $\delta(\cdot,\cdot)$. Note that $c(\vec{s})$ depends on the replicated configuration, i. e., on the vector $\vec{\sigma}$ having nN components. These fractions are restricted by the normalization $\sum_{\vec{s}} c(\vec{s}) = 1$, so there are $2^n - 1$ different order-parameter components (compared to a single one in the Weiss model). They take values in $\{0, 1/N, 2/N, \ldots, 1\}$. Using this definition, the interaction term can be expressed as an order-parameter function,

$$\begin{aligned} \sum_{ij} e^{\beta \vec{\sigma}_i^a \cdot \vec{\sigma}_j^a} &= \sum_{ij} \sum_{\vec{s}_1} \delta_{\vec{s}_1,\sigma_i} \sum_{\vec{s}_2} \delta_{\vec{s}_2,\sigma_j} e^{\beta \vec{s}_1 \cdot \vec{s}_2} \\ &= N^2 \sum_{\vec{s}_1,\vec{s}_2} c(\vec{s}_1) c(\vec{s}_2) e^{\beta \vec{s}_1 \cdot \vec{s}_2} \,. \end{aligned} \tag{5.35}$$

where the last line results from an interchange of the $\vec{s}_{1,2}$ and the (i,j)-summations. The replicated and disorder-averaged partition function can thus be rewritten as

$$\overline{Z(J)^n} = \sum_{\{\sigma_i^a\}} \sum_{\{c(\cdot)\}} \prod_{\vec{s}} \delta\left(c(\vec{s}), \frac{1}{N} \sum_i \delta_{\vec{s},\vec{\sigma}_i}\right) \exp\left\{-\frac{cN}{2} + \frac{cN}{2} \sum_{\vec{s}_1,\vec{s}_2} c(\vec{s}_1) c(\vec{s}_2) e^{\beta \vec{s}_1 \cdot \vec{s}_2}\right\} \tag{5.36}$$

because the $\delta(\cdot,\cdot)$ term picks exactly the $c(\vec{\sigma})$ corresponding to the configuration $\{\vec{\sigma}_i\}$ under consideration. The sum over $c(\vec{s})$ runs over the allowed values mentioned above. The dependency on the local replicated spins $\vec{\sigma}_i$ is now completely reduced to the definition of the order parameter. We interchange the sums in the last equation and have to calculate the number of replicated configurations $\{\vec{\sigma}_i\}$ leading to a given set of $c(\vec{s})$:

$$\begin{aligned} \sum_{\{\sigma_i^a\}} \prod_{\vec{s}} \delta\left(c(\vec{s}), \frac{1}{N} \sum_i \delta_{\vec{s},\vec{\sigma}_i}\right) &= \frac{N!}{\prod_{\vec{s}} (c(\vec{s})N)!} \\ &= \exp\left\{-N \sum_{\vec{s}} c(\vec{s}) \ln c(\vec{s}) + \mathcal{O}(\ln N)\right\}. \end{aligned} \tag{5.37}$$

The first step results from the fact, that, for every $\vec{s}$, the $c(\vec{s})N$ sites having spin equal to $\vec{s}$ can be arbitrarily selected among the N sites. The resulting multinomial coefficient is a direct generalization of the well-known binomial coefficient. The step from the first to the second

line uses Stirling's formula, in complete analogy to the calculation for the Weiss model in Eq. (5.15). Replacing the sum over the order parameters by an integration for large N, still considering a fixed finite number n of replicas, we find

$$\begin{aligned}\overline{Z(J)^n} \\ &= \int_{\mathcal{S}} \prod_{\vec{s}} dc(\vec{s}) \exp\left\{-N\left[\frac{c}{2} - \frac{c}{2}\sum_{\vec{s}_1,\vec{s}_2} c(\vec{s}_1)c(\vec{s}_2)e^{\beta\vec{s}_1\cdot\vec{s}_2} + \sum_{\vec{s}} c(\vec{s})\ln c(\vec{s})\right]\right\} \\ &\equiv \int_{\mathcal{S}} \prod_{\vec{s}} dc(\vec{s}) \exp\left\{-\beta N f_n[c(\vec{s})]\right\} \qquad (5.38)\end{aligned}$$

where the integration domain is given by $\mathcal{S} = \{c(\vec{s}) \mid \forall\vec{s} : 0 \le c(\vec{s}) \le 1,\ \sum_{\vec{s}} c(\vec{s}) = 1\}$. The sum over all configurations is performed, only a finite-dimensional integration over a 2^n-dimensional unit sphere remains to be done. This integration can be achieved using the saddle-point method introduced before, because there is a large multiplier in the exponential. The saddle point is, however, restricted by the normalization condition which can be taken into account using a Lagrangian multiplier:

$$0 = \frac{\partial}{\partial c(\vec{s})}\left(\beta f_n[c(\vec{s})] + \lambda\left\{\sum_{\vec{s}} c(\vec{s}) - 1\right\}\right) . \qquad (5.39)$$

Using $\partial c(\vec{s}_1)/\partial c(\vec{s}_2) = \delta_{\vec{s}_1,\vec{s}_2}$ for the 2^n different variables $c(\vec{s})$, we find the following saddle-point equation,

$$0 = -c\sum_{\vec{s}_1} c(\vec{s}_1)e^{\beta\vec{s}_1\cdot\vec{s}} + \ln c(\vec{s}) + 1 + \lambda \qquad (5.40)$$

which can easily be rewritten as

$$c(\vec{s}) = \exp\left\{-1 - \lambda + c\sum_{\vec{s}_1} c(\vec{s}_1)e^{\beta\vec{s}_1\cdot\vec{s}}\right\} . \qquad (5.41)$$

These 2^n equations are completed by the normalization constraint

$$\sum_{\vec{s}} c(\vec{s}) = 1 . \qquad (5.42)$$

Before solving equations (5.41,5.42) in such a way that the replica limit $n \to 0$ can be performed, we want to first discuss the symmetries of the problem.

Symmetries

There are two different kinds of symmetry present in the replicated Hamiltonian.

- *Spin-flip symmetry*: By construction, the system is spin-flip symmetric. This symmetry is valid independently for every replica. If, for any fixed $a \in \{1, \ldots, n\}$, we change $(\sigma_1^a, \ldots, \sigma_N^a)$ to $(-\sigma_1^a, \ldots, -\sigma_N^a)$, the energy of the ath replica is unchanged.
- *Replica symmetry*: All the n replicas are equivalent copies of the original system, their numeration is thus arbitrary. This means that the replicated Hamiltonian is obviously invariant under any permutation $(1, \ldots, n) \mapsto (\pi(1), \ldots, \pi(n))$ of the replicas, where π denotes an arbitrary element of the permutation group $\mathcal{S}_n$ of n elements.

As usual, we would expect that for high temperatures all symmetries are also unbroken at the order parameter level. On the other hand, we know from the Weiss model, that ferromagnetic behavior is related to a spontaneous breaking of the spin-flip symmetry. As we will see below, this is also sufficient for the random-graph Ising model. The replica symmetry will be unbroken in this model.

The fully symmetric solution: paramagnetism

Obviously, the only fully symmetric version of $c(\vec{s})$ is

$$c(\vec{s}) = \frac{1}{2^n} \tag{5.43}$$

because every $\vec{s}$ can be transformed to any other one by, at most, n flips of single components of $\vec{s}$. It is a solution of the saddle-point equation: Plugging (5.43) into the right-hand side of the saddle-point equation (5.41), it becomes independent of $\vec{s}$. This is consistent, because the left-hand side is here independent of $\vec{s}$ as well. The correct value of the Lagrangian multiplier is fixed by the normalization which leads to Eq. (5.43). Obviously, this solution describes a paramagnet because, for each replica, positive and negative spin orientations appear with the same probability. Solution (5.43) can be plugged into $f_n[c(\vec{s})]$, and easily evaluated, which we do not need to do here. n appears now as a mere parameter, hence the replica limit $n \to 0$ needed in Eq. (5.31) can be easily achieved.

Breaking the spin-flip symmetry

In order to enter the ferromagnetic phase, we have to break the spin-flip symmetry of the order parameter (in analogy with what we have done in the Weiss model when we considered non-zero magnetization). On the other hand, we will use an ansatz which is replica symmetric. The only non-trivial replica-symmetric combination which can be constructed out of the components of $\vec{s}$ is $\sum_a s^a$. All other replica-symmetric expressions are functions of this sum because $\vec{s}$ has only binary components. So we conclude that a replica-symmetric $c(\vec{s})$ can

only be a function $c(\sum_a s^a)$ of this replica invariant. The Laplace transform $\tilde{c}(s)$ of a function $c(h)$ is defined via $\tilde{c}(s) = \int dh\, \tilde{c}(h) e^{-hs}$. Similarly, for suitably chosen function $P(h)$, the function $c\left(\sum_a s^a\right)$ can be represented as a kind of Laplace transform via

$$\begin{aligned} c(\vec{s}) &= c\left(\sum_a s^a\right) \\ &= \int dh\, P(h)\, \frac{e^{\beta h \sum_a s^a}}{(2\cosh \beta h)^n}\,, \end{aligned} \tag{5.44}$$

where the factor $\cosh \beta h$ is included for convenience. Because of the normalization of $c(\vec{s})$, the function $P(h)$ also has to be normalized:

$$\begin{aligned} 1 &= \sum_{\vec{s}} c(\vec{s}) = \int dh\, P(h)\, \frac{\sum_{\vec{s}} e^{\beta h \sum_a s^a}}{(2\cosh \beta h)^n} \\ &= \int dh\, P(h)\, \frac{\prod_a \left(\sum_{s^a = \pm 1} e^{\beta h s^a}\right)}{(2\cosh \beta h)^n} \\ &= \int dh\, P(h)\, \frac{\prod_a 2\cosh \beta h}{(2\cosh \beta h)^n} = \int dh\, P(h)\,. \end{aligned} \tag{5.45}$$

It can thus be interpreted as a distribution of *effective local fields*. Imagine, for a moment, an isolated replicated spin $\vec{s}$ in an external field h. The corresponding energy would be $-h\sum_a s^a$, and would lead to a Gibbs weight $\exp\{\beta h \sum_a s^a\}$. Together with the proper normalization, this is nothing other that the fraction in the last line of Eq. (5.44).

From Eq. (5.44) many statistical properties of the system at inverse temperature β can be read off. We can, e. g., calculate the average magnetization

$$\begin{aligned} m &= \lim_{n\to 0} \sum_{\vec{s}} s^1 c(\vec{s}) \\ &= \int dh\, P(h)\, \tanh(\beta h)\,, \end{aligned} \tag{5.46}$$

note that after performing the sum $\sum_{\vec{s}}$ in the first line, the remainder becomes independent of n, hence the $\lim_{n\to 0}$ just drops out. This also provides a description of how $P(h)$ can be determined in a numerical experiment. If the system is simulated by means of, e. g., a Monte Carlo simulation, one can measure the local magnetization m_i of each spin as a time average of σ_i. This magnetization is related to an effective field via $m_i = \tanh(\beta h_i)$ which fluctuates from site to site due to the inhomogeneities of the random graph. The distribution $P(h)$ can be found as the histogram of all the h_i.

In order also to determine analytically $P(h)$, we plug ansatz (5.44) into the saddle-point equation (5.41). Using the abbreviation $\mu = \beta \sum_a s^a$, we get

$$\begin{aligned}
&\int dh\ P(h)\ \frac{e^{h\mu}}{(2\cosh\beta h)^n} \\
&= \exp\left\{-1-\lambda+c\int dh\ P(h)\frac{\sum_{\vec{s}_1} e^{\beta\sum_a (s^a+h)s_1^a}}{(2\cosh\beta h)^n}\right\} \\
&= \exp\left\{-1-\lambda+c\int dh\ P(h)\frac{\prod_a \left(\sum_{s_1^a} e^{\beta(s^a+h)s_1^a}\right)}{(2\cosh\beta h)^n}\right\} \qquad (5.47)\\
&= \exp\left\{-1-\lambda+c\int dh\ P(h)\frac{(2\cosh\beta(h+1))^{\frac{n+\mu/\beta}{2}}(2\cosh\beta(h-1))^{\frac{n-\mu/\beta}{2}}}{(2\cosh\beta h)^n}\right\}
\end{aligned}$$

where we have used that the factors over the replicas $a = 1, \ldots, n$ in the exponential depend only on $s^a = +1$ or $s^a = -1$. Finally we have reached a point where the limit $n \to 0$ can be performed, we thus find

$$\int dh\ P(h)\ e^{h\mu} = \exp\left\{-c + c\int dh\ P(h)\left[\frac{\cosh\beta(h+1)}{\cosh\beta(h-1)}\right]^{\frac{\mu}{2\beta}}\right\}. \qquad (5.48)$$

We have already used the correct value $\lambda = -1 - c$ of the Lagrangian multiplier to ensure normalization of $P(h)$, as can be checked by setting $\mu = 0$. This equation has to be solved for arbitrary μ. If we expand the exponential on the right-hand side, and compare coefficients of exponentials in μ, we find

$$P(h) = \sum_{d=0}^{\infty} e^{-c}\frac{c^d}{d!}\int dh_1 \cdots dh_d\ P(h_1)\cdots P(h_d)\ \delta\left(h - \sum_{l=1}^{d}\frac{1}{2\beta}\ln\frac{\cosh\beta(h_l+1)}{\cosh\beta(h_l-1)}\right), \qquad (5.49)$$

as can be checked most easily by plugging Eq. (5.49) into the left-hand side of Eq. (5.47). This equation has a very nice form, it can be seen as an iterative prescription for $P(h)$. Plugging some test function into the right-hand side, a new function is generated. The solution of Eq. (5.49) is a fixed point of this procedure.

The phase diagram

However, we still have to solve this non-linear integral equation. First we observe that $P(h) = \delta(h)$ is always a solution. If all fields vanish, the local magnetizations also vanish. The solution thus describes the paramagnetic phase. It can be seen also that, if we plug it into Eq. (5.44), we get back to Eq. (5.43).

In order to observe the onset of a non-trivial, ferromagnetic solution of Eq. (5.49), we use the iteration mentioned before, and perform a *linear stability analysis* of the paramagnetic

solution. On the right-hand side, we plug in some function $P_0(h)$ allowing for small, non-zero fields, which leads also to a small, and non-zero mean $0 < \varepsilon_0 = \int dh\ h\ P_0(h) \ll 1$, and observe how this mean evolves under iteration. Denoting the result of the first iteration by $P_1(h)$, we calculate the new mean

$$\begin{aligned}
\varepsilon_1 &= \int dh\ h\ P_1(h) \\
&= \sum_{d=0}^{\infty} e^{-c} \frac{c^d}{d!} \int dh_1 \cdots dh_d\ P_0(h_1) \cdots P_0(h_d)\ \left(\sum_{l=1}^{d} \frac{1}{2\beta} \ln \frac{\cosh \beta(h_l + 1)}{\cosh \beta(h_l - 1)} \right) \\
&= \int dh\ P_0(h)\ \frac{1}{2\beta} \ln \frac{\cosh \beta(h+1)}{\cosh \beta(h-1)} \left(\sum_{d=0}^{\infty} e^{-c} \frac{c^d}{d!} d \right)
\end{aligned} \tag{5.50}$$

where we have interchanged the sum and the integrations in the last line, and we have used the normalization of $P_0(h)$. The summation can be performed giving a factor c. The integrand can be linearized in h since the fields contributing to $P_0(h)$ are small, i. e., when performing a Taylor expansion around $h = 0$ one obtains:

$$\frac{1}{2\beta} \ln \frac{\cosh \beta(h+1)}{\cosh \beta(h-1)} = h\ \tanh \beta + \mathcal{O}(h^2)\ . \tag{5.51}$$

In linear approximation we thus get

$$\varepsilon_1 = (c \tanh \beta)\ \varepsilon_0\ . \tag{5.52}$$

For

$$c \tanh \beta < 1 \tag{5.53}$$

the iteration thus brings us back to the paramagnetic solution, i. e., the resulting field distribution $P(h) = \delta(h)$ is locally stable.

Equality in (5.53) defines thus the critical line $T_c(c)$ above which the model behaves as a paramagnet, see Fig. 5.5. Below this line, the paramagnetic solution becomes locally unstable, a small perturbation evolves towards some finite global magnetization. The system is ferromagnetic.

There are two important observations:

- For $c < 1$, the system is always paramagnetic. Ferromagnetic low-temperature behavior is found only beyond $c = 1$. This confirms exactly what we were expecting from our knowledge about the random-graph structure below its percolation point.
- For $c \gg 1$, we have $T_c(c) \simeq c$. This is consistent with what we found in the Weiss model. There, the critical temperature was found to be $T_c = 1$, but the Hamiltonian was rescaled by a factor $1/N$. The only combination of temperature and Hamiltonian entering the partition function i s βH, so we could shift the scaling factor from the Hamiltonian to the temperature, finding thus a critical temperature equal to N. This equals, up to a correction of one, the vertex degree of the complete graph.

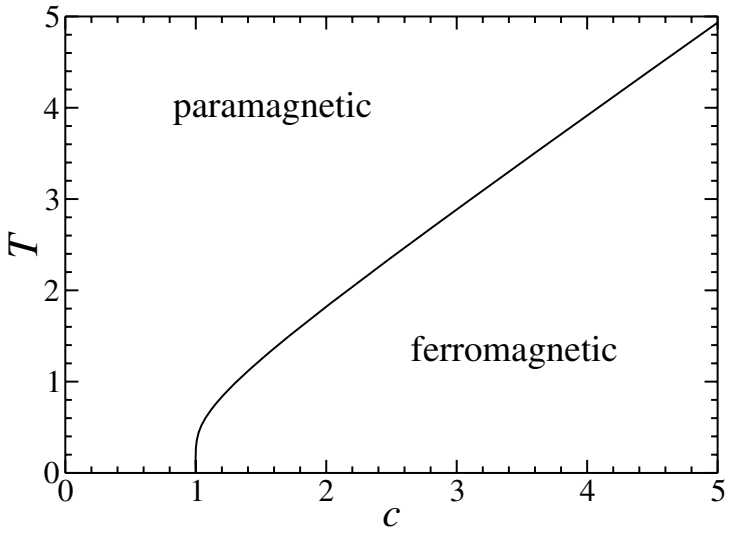


Figure 5.5: Phase diagram of the Ising model on a random graph.

The population-dynamical algorithm

For $0 < T < T_c(c)$, the solution can be found only numerically, but there is an efficient *population-dynamical algorithm* to do this [18]. Instead of working with $P(h)$ directly, we represent the effective-field distribution by a large population $\{h_1, \ldots, h_M\}$ of $M \gg 1$ fields randomly drawn from $P(h)$. Running the algorithm, this population is first initialized randomly, and then Eq. (5.49) is used to iteratively replace fields inside the population, until convergence is reached.

The action of the algorithm is summarized in the following pseudo code:

algorithm PopDyn($h_1, \ldots, h_M$)
begin
  do
    draw d from Poisson distribution $(e^{-c}c^d/d!)$;
    select randomly $d+1$ indices $i, i_1, \ldots, i_d \in \{1, \ldots, M\}$;
    $h_i := \sum_{l=1}^{d} \frac{1}{2\beta} \ln \frac{\cosh \beta(h_{i_l}+1)}{\cosh \beta(h_{i_l}-1)}$;
  while (not converged)
  return $(h_1, \ldots, h_M)$;
end

The algorithm is of stochastic nature, thus convergence does not mean that the population remains invariant, but its statistical properties, in particular its histogram, become constant up to fluctuations. The histogram finally gives an approximate solution of Eq. (5.49). It becomes

increasingly precise if the population size M is increased, and if the histogram is averaged over its stochastic fluctuations.

Zero-temperature behavior

Finally, at $T = 0$ (i. e., $\beta \to \infty$) the solution can again be determined analytically. Using the fact that $2\cosh\beta h \simeq e^{\beta|h|}$ for large β, we can calculate the limit

$$\kappa(h) := \lim_{\beta\to\infty} \frac{1}{2\beta} \ln \frac{\cosh\beta(h+1)}{\cosh\beta(h-1)} = \begin{cases} 1 & \text{if} \quad 1 < h \\ h & \text{if} \quad -1 < h < 1 \\ -1 & \text{if} \quad h < -1 \end{cases} , \tag{5.54}$$

and Eq. (5.49) simplifies to

$$P(h) = \sum_{d=0}^{\infty} e^{-c} \frac{c^d}{d!} \int dh_1 \cdots dh_d \; P(h_1) \cdots P(h_d) \; \delta\left(h - \sum_{l=1}^{d} \kappa(h_l)\right) . \tag{5.55}$$

The solution of this equation is further simplified by the fact that, in the original system, a spin flip causes an energy difference which is an integer multiple of 2. If a spin exposed to a field h is flipped, its energy difference is $\pm 2h$. We thus expect the fields to take only integer values! In addition, using our physical intuition resulting from the Weiss model, we do not expect negative and positive fields (i. e., negative and positive magnetizations) to coexist. According to our discussion in the beginning, spins belonging to finite connected components of the graph G are paramagnetic, with vanishing fields. Spins from the giant component have the same sign as their magnetizations, i. e., the same sign as their effective fields. We thus use the ansatz

$$P(h) = \sum_{l=0}^{\infty} r_l \delta(h - l) \tag{5.56}$$

with still unknown coefficients r_l. Now the solution of the saddle-point equation becomes very easy. The field $h = 0$ is generated if and only if all neighboring fields are also zero, i. e., contributions to r_0 are

$$\begin{aligned} r_0 &= \sum_{d=0}^{\infty} e^{-c} \frac{c^d}{d!} r_0^d \\ &= e^{-c(1-r_0)} . \end{aligned} \tag{5.57}$$

As mentioned above, field $h = 0$ corresponds to paramagnetic spins. All other spins have positive integer fields, i. e., their ground state magnetization $\lim_{\beta\to\infty} \tanh(\beta h)$ tends to one. These spins behave ferromagnetically. Summing up the fractions for all positive fields, we get the fraction γ of ferromagnetic spins:

$$\gamma = \sum_{l=1}^{\infty} r_l = 1 - r_0 . \tag{5.58}$$

According to Eq. (5.57), this quantity is determined by

$$1 - \gamma = e^{-c\gamma} \,. \tag{5.59}$$

This is, however, exactly the equation which determines the size of the giant component of the random graph. For $c < 1$, the only solution is $\gamma = 0$, i. e., we find again that the model behaves paramagnetically down to zero temperature. For $c > 1$, there is a non-trivial solution, which in fact describes the stable saddle point.

At this point we stop the replica analysis. In principle one could plug the solution into the free-energy functional and analyze thermodynamic quantities like the average energy, the entropy of ground states, etc. Instead of doing this, we will now discuss another method, which allows one to describe the thermodynamic behavior of the Ising model on random graphs.

5.4.4 The Bethe–Peierls approach

This other method is the so-called Bethe–Peierls iterative method [20, 21], and can be seen as the replica symmetric version of the cavity approach for finite-degree models [18]. The latter allows also for the treatment of replica-symmetry broken systems, and therefore forms an alternative to the replica approach.

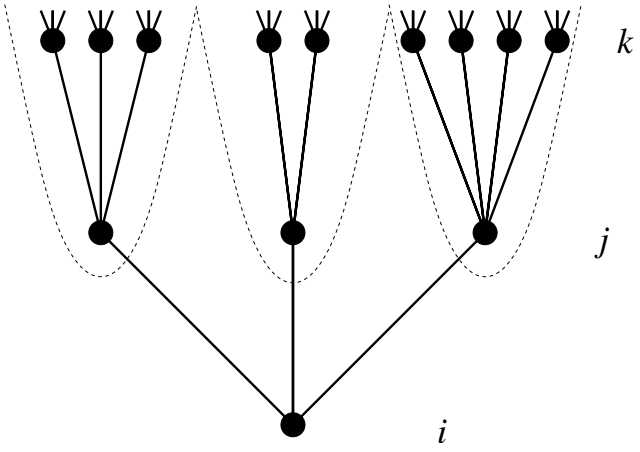


Figure 5.6: Iteration tree for the calculation of the partition function. The subtrees rooted in all $j \in N(i)$ are marked by the dotted lines.

The approach for the Ising model (5.28) on a random graph is mainly based on the *locally tree-like* structure of the underlying graph. The latter was discussed in detail in Sec. 3.3.3: Almost all loops are of length $\mathcal{O}(\ln N)$ and therefore diverge in the thermodynamic limit $N \to \infty$.

To derive the order parameter equations, we assume for a moment that the graph is really a tree. In this case, the partition function can be calculated via a simple iterative scheme. Let

us select an arbitrary vertex $i \in V$, and introduce the restricted partition function $Z_i(\sigma_i)$ with the spin of vertex i fixed to value σ_i,

$$Z_i(\sigma_i) = \sum_{\{\sigma_l, l\neq i\}} e^{-\beta H_J(\sigma_1,\ldots,\sigma_N)} \,. \tag{5.60}$$

It can be determined by the restricted partition functions $Z_{j|i}(\sigma_j)$ of the subtrees "above" i, i. e., the subtrees which are rooted in the neighbors j of i, but do not contain vertex i. In Fig. 5.6, they are given as the interior of the dashed lines. We find

$$Z_i(\sigma_i) = \sum_{\{\sigma_j, j\in N(i)\}} e^{\beta\sigma_i \sum_{j\in N(i)} \sigma_j} \prod_{j\in N(i)} Z_{j|i}(\sigma_j) \,. \tag{5.61}$$

The set $N(i)$ contains all neighbors of i. In this formula we have explicitly used the assumption that the model is defined on a tree, only in this case, all $j \in N(i)$ are roots of pairwise disconnected subtrees, and the total partition function factorizes in j. The subtree partition functions $Z_{j|i}(\sigma_j)$ now depend on the subtrees "above" the j, i. e., on the second neighbors of i:

$$Z_{j|i}(\sigma_j) = \sum_{\{\sigma_k, k\in N(j)\setminus i\}} e^{\beta\sigma_j \sum_{k\in N(j)\setminus i} \sigma_k} \prod_{k\in N(j)\setminus i} Z_{k|j}(\sigma_k) \,. \tag{5.62}$$

Continuing like this, we can also go to the third, fourth, etc., neighbors of i. In order to avoid writing down an infinite hierarchy, we next introduce *cavity fields* $h_{j|i}$ via

$$\begin{aligned}
h_{j|i} &\equiv \frac{1}{2\beta} \ln \frac{Z_{j|i}(+1)}{Z_{j|i}(-1)} \\
&= \frac{1}{2\beta} \ln \frac{\sum_{\{\sigma_k, k\in N(j)\setminus i\}} e^{+\beta \sum_{k\in N(j)\setminus i} \sigma_k} \prod_{k\in N(j)\setminus i} Z_{k|j}(\sigma_k)}{\sum_{\{\sigma_k, k\in N(j)\setminus i\}} e^{-\beta \sum_{k\in N(j)\setminus i} \sigma_k} \prod_{k\in N(j)\setminus i} Z_{k|j}(\sigma_k)} \\
&= \frac{1}{2\beta} \ln \prod_{k\in N(j)\setminus i} \frac{\sum_{\sigma_k} e^{+\beta\sigma_k} Z_{k|j}(\sigma_k)}{\sum_{\sigma_k} e^{-\beta\sigma_k} Z_{k|j}(\sigma_k)} \\
&= \frac{1}{2\beta} \ln \prod_{k\in N(j)\setminus i} \frac{e^{+\beta} Z_{k|j}(+1) + e^{-\beta} Z_{k|j}(-1)}{e^{-\beta} Z_{k|j}(+1) + e^{+\beta} Z_{k|j}(-1)} \\
&= \frac{1}{2\beta} \ln \prod_{k\in N(j)\setminus i} \frac{e^{+\beta} e^{-\beta h_{k|j}} \frac{Z_{k|j}(+1)}{Z_{k|j}(-1)} + e^{-\beta(h_{k|j}+1)}}{e^{-\beta} e^{-\beta h_{k|j}} \frac{Z_{k|j}(+1)}{Z_{k|j}(-1)} + e^{-\beta(h_{k|j}-1)}} \\
&= \frac{1}{2\beta} \sum_{k\in N(j)\setminus i} \ln \frac{\cosh\beta(h_{k|j}+1)}{\cosh\beta(h_{k|j}-1)} \,,
\end{aligned} \tag{5.63}$$

which, for every rooted subtree, can thus be calculated by the cavity fields of the neighbors inside the subtree. To go from the fourth to the fifth line, we have divided both the numerator and denominator inside the logarithm by $\prod_{k\in N(j)\setminus i} e^{\beta h_{k|j}} Z_{k|j}(-1)$. In complete analogy,

we introduce the local *effective field*

$$\begin{aligned} h_i &= \frac{1}{2\beta} \ln \frac{Z_i(+1)}{Z_i(-1)} \\ &= \frac{1}{2\beta} \sum_{j \in N(i)} \ln \frac{\cosh \beta(h_{j|i} + 1)}{\cosh \beta(h_{j|i} - 1)} \,, \end{aligned} \tag{5.64}$$

where now the sum runs over *all* neighbors of vertex i. This effective field has, from its definition in the first line, a very simple interpretation. The marginal probability $p(\sigma_i)$ for the single spin i to have orientation σ_i is given by

$$p(\sigma_i) = e^{\beta h_i \sigma_i} / (2 \cosh \beta h_i) \,. \tag{5.65}$$

This can be verified easily by computing $p(+1)$ and $p(-1)$ and using the definition of h_i. Consequently, its local magnetization reads $m_i = \tanh \beta h_i$. Hence, spin i behaves like a single isolated spin in an external field h_i. In complete analogy, for a spin j, without a neighbor i, the probability of having an orientation σ_j is

$$p_{j|i}(\sigma_j) = e^{\beta h_{j|i} \sigma_i} / (2 \cosh \beta h_{j|i}) \,. \tag{5.66}$$

So far we have considered trees. Our objective is, however, to describe random graphs. The subdivision into pairwise disconnected subtrees, as in Eq. (5.61), is therefore not correct. The restricted partition function $Z_i(\sigma_i)$ depends on the joint partition function $Z_{N(i)|i}(\{\sigma_j, j \in N(i)\})$ of all neighbors of i, with i eliminated from the graph. We introduce therefore the *cavity graph* G_i by eliminating i and all its incident edges from the original graph G, or, in more mathematical terms, G_i is the subgraph of G induced by the vertex set $V \setminus i$ containing all vertices but i. We exploit the locally tree-like structure of the graph. Almost all loops have a length of $\mathcal{O}(\ln N)$ in G. Consider now any two vertices from $N(i)$; being second neighbors in the full graph, they are almost surely distant in the cavity graph! Inside one thermodynamic state two-spin correlations decay exponentially fast with their distance, so the two considered spins are therefore practically uncorrelated as far as $N \gg 1$ is considered. So even if the partition function does not factorize trivially, the joint marginal distribution of these spins factorizes. Equations (5.63) and (5.64) are therefore valid inside one state in the thermodynamic limit.

These equations therefore describe a self-consistent system of equations first for determining the cavity fields associated to each *edge* (and each direction of the edge), and then, on this basis, also the physical fields. One could solve these equations numerically for a given graph. For this purpose, one calculates first the cavity fields $h_{j|i}$ for all edge pairs (i, j) self-consistently by starting with some values $\{h_{j|i}\}$ and iterating Eqs (5.63) till convergence. Finally one obtains the effective fields h_i from Eqs (5.64), which determine the complete physical behavior.

Here, however, we return to random graphs, and characterize typical, i. e., ensemble-averaged quantities. We introduce the *cavity-field distribution*

$$P_{\text{cav}}(h) = \frac{1}{cN} \sum_{\{i,j\} \in E} \left[\delta(h - h_{i|j}) + \delta(h - h_{j|i}) \right] \tag{5.67}$$

as the histogram over all cavity fields of the graph. Note that each edge contributes two fields, according to which vertex is the one associated to the cavity field, and which one is associated with the cavity. We furthermore introduce the analogous *effective-field distribution*

$$P(h) = \frac{1}{N} \sum_{i \in V} \delta(h - h_i) \tag{5.68}$$

by considering the effective fields of all vertices. Note that the last distribution has already been considered in Eq. (5.44) in the context of the replica approach.

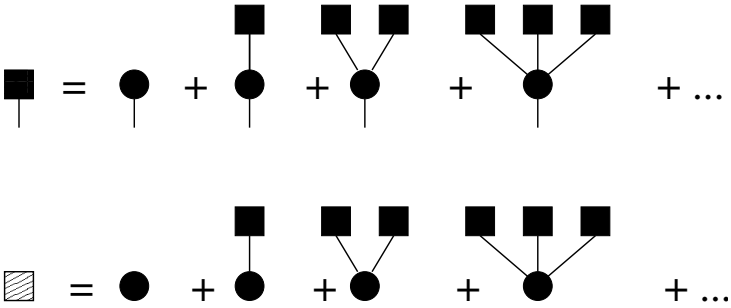


Figure 5.7: Graphical solution of the cavity-field distribution (represented by the black square with the half-edge) and the physical effective-field distribution (represented by the shaded square). Each diagram has to be read from its top to its bottom. At the nodes of the diagrams, the induced fields have to be summed up.

The cavity-field distribution can be determined from a self-consistent equation which is represented graphically in the first line of Fig. 5.7. A cavity field $h_{j|i}$ is generated by the cavity fields of d neighbors, where d is the excess degree of vertex j with respect to edge $\{i, j\}$, i. e., d is distributed according to the modified distribution $q_{d+1} = (d+1)p_{d+1}/c$ introduced in Eq. (3.13). We thus find, in analogy to Eq. (3.15),

$$\begin{aligned} P_{\mathrm{cav}}(h) &= \sum_{d=0}^{\infty} q_{d+1} \int dh_1 \cdots dh_d \, P_{\mathrm{cav}}(h_1) \cdots P_{\mathrm{cav}}(h_d) \\ &\qquad \times \delta \left(h - \sum_{l=1}^{d} \frac{1}{2\beta} \ln \frac{\cosh \beta(h_l + 1)}{\cosh \beta(h_l - 1)} \right) \\ &= \sum_{d=0}^{\infty} e^{-c} \frac{c^d}{d!} \int dh_1 \cdots dh_d \, P_{\mathrm{cav}}(h_1) \cdots P_{\mathrm{cav}}(h_d) \\ &\qquad \times \delta \left(h - \sum_{l=1}^{d} \frac{1}{2\beta} \ln \frac{\cosh \beta(h_l + 1)}{\cosh \beta(h_l - 1)} \right) . \end{aligned} \tag{5.69}$$

Note that this equation is identical to Eq. (5.49), resulting from the replica approach, but with $P_{\mathrm{cav}}(h)$ and $P(h)$ exchanged. Given the different interpretation of these two distributions,

this coincidence may seem obscure. It becomes, however, immediately clear if we write down the equation for $P(h)$ as it results from $P_{\rm cav}(h)$, in analogy to Eq. (3.16),

$$\begin{aligned} P(h) &= \sum_{d=0}^{\infty} p_d \int dh_1 \cdots dh_d \, P_{\rm cav}(h_1) \cdots P_{\rm cav}(h_d) \\ &\qquad \times \delta \left(h - \sum_{l=1}^{d} \frac{1}{2\beta} \ln \frac{\cosh \beta(h_l + 1)}{\cosh \beta(h_l - 1)} \right) \\ &= \sum_{d=0}^{\infty} e^{-c} \frac{c^d}{d!} \int dh_1 \cdots dh_d \, P_{\rm cav}(h_1) \cdots P_{\rm cav}(h_d) \\ &\qquad \times \delta \left(h - \sum_{l=1}^{d} \frac{1}{2\beta} \ln \frac{\cosh \beta(h_l + 1)}{\cosh \beta(h_l - 1)} \right) . \end{aligned} \tag{5.70}$$

Due to a particularity of random graphs, namely the equality $q_{d+1} = p_d$ resulting from the Poissonian degree distribution, we thus find $P(h) = P_{\rm cav}(h)$, even if the fields on single vertices do not coincide. This accidental equality also allows for a simpler interpretation of Eq. (5.49) which would be much less evident, without the subtle distinction between cavity fields and effective fields.

The possible solutions of these equations have already been discussed in Sec. 5.4.3. Here we add another aspect, namely the calculation of global quantities as energy and free energy from the order parameter.

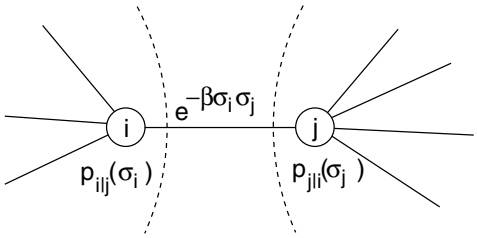


Figure 5.8: An edge in the graph. The joint probability for spins i and j to have orientations σ_i and σ_j is proportional to $p_{i|j}(\sigma_i) e^{-\beta\sigma_i\sigma_j} p_{j|i}(\sigma_j)$.

The average energy is very simple, since the Hamiltonian is given by a sum over pair interactions, which can be averaged one by one. Assuming still a tree structure (or decorrelation through long loops), for each pair i, j, the joint probability that spin i has orientation σ_i *and* that spin j has orientation σ_j is the product of the probability $p_{i|j}(\sigma_i)$ that spin i has orientation σ_i without the presence of spin j, times the probability $p_{|i}(\sigma_j)$ that spin j has orientation σ_j without the presence of spin i, times $\frac{1}{K} e^{-\beta\sigma_i\sigma_j}$ (K being a suitably chosen normalization constant), see also Fig. 5.8. Using Eq. (5.66), i. e., via $p_{j|i}(\sigma_j) \sim e^{-\beta h_{j|i}\sigma_i}$ and using

normalization, we obtain

$$\begin{aligned}\langle H_J\rangle_T &= -\sum_{i<j} J_{ij}\,\langle\sigma_i\sigma_j\rangle_T \\ &= -\sum_{i<j} J_{ij}\sum_{\sigma_i,\sigma_j}\sigma_i\sigma_j\frac{1}{K}e^{-\beta\sigma_i\sigma_j}p_{i|j}(\sigma_i)p_{j|i}(\sigma_j) \\ &= -\sum_{i<j} J_{ij}\frac{\sum_{\sigma_i,\sigma_j}\sigma_i\sigma_j\exp\{\beta(\sigma_i\sigma_j+h_{i|j}\sigma_i+h_{j|i}\sigma_j)\}}{\sum_{\sigma_i,\sigma_j}\exp\{\beta(\sigma_i\sigma_j+h_{i|j}\sigma_i+h_{j|i}\sigma_j)\}}\,, \end{aligned} \tag{5.71}$$

i. e., the influence of the graph on one single interaction term is described by the cavity fields. After having averaged over the graph ensemble $\mathcal{G}(N,c/N)$, the energy density therefore reads

$$\begin{aligned} e &= \frac{\overline{\langle H_J\rangle_T}}{N} \qquad (5.72)\\ &= -\frac{c}{2}\int dh_1 P_{\rm cav}(h_1)dh_2P_{\rm cav}(h_2)\frac{\sum_{\sigma_1,\sigma_2}\sigma_1\sigma_2\exp\{\beta(\sigma_1\sigma_2+h_1\sigma_1+h_2\sigma_2)\}}{\sum_{\sigma_1,\sigma_2}\exp\{\beta(\sigma_1\sigma_2+h_1\sigma_1+h_2\sigma_2)\}}\,. \end{aligned}$$

The determination of the free-energy density is a bit more involved, it reads

$$F_J=\sum_{i<j}J_{ij}F_{ij}-\sum_i(\deg(\mathrm{i})-1)\,F_i \tag{5.73}$$

with the site contributions

$$e^{-\beta F_i}=\sum_{\sigma_i}e^{\beta h_i\sigma_i}=2\cosh\beta h_i \tag{5.74}$$

and the link contributions

$$e^{-\beta F_{ij}}=\sum_{\sigma_i,\sigma_j}\exp\{\beta(\sigma_i\sigma_j+h_{i|j}\sigma_i+h_{j|i}\sigma_j)\}\,. \tag{5.75}$$

This formula can be easily verified on a tree. Here we give only an intuitive justification for its specific form: Summing the free-energy contributions of all single edges, we have counted every vertex i exactly $\deg(i)$ times. This over-counting is compensated for in the second part of Eq. (5.73).

After averaging over the graph ensemble, we therefore find

$$f=\frac{\overline{F_J}}{N}=\frac{c}{2}f_{\rm link}-f_{\rm site} \tag{5.76}$$

with

$$\begin{aligned} &e^{-\beta f_{\rm site}}= \\ &2\sum_{d=0}^{\infty}e^{-c}\frac{(d-1)c^d}{d!}\int\prod_{l=1}^{d}[dh_l\;P_{\rm cav}(h_l)]\cosh\left\{\beta\sum_{l=1}^{d}\frac{1}{2\beta}\ln\frac{\cosh\beta(h_l+1)}{\cosh\beta(h_l-1)}\right\} \end{aligned} \tag{5.77}$$

and

$$e^{-\beta f_{\rm link}} = \int dh_1 P_{\rm cav}(h_1) dh_2 P_{\rm cav}(h_2) \sum_{\sigma_1,\sigma_2} \exp\{\beta(\sigma_1\sigma_2 + h_1\sigma_1 + h_2\sigma_2)\} \ . \quad (5.78)$$

The difference between the energy and the free energy finally gives the entropy,

$$s = \beta(e - f) \quad (5.79)$$

which, for $T = 0$, equals the logarithm of the average number of connected components of a $\mathcal{G}(N, c/N)$ graph, divided by the graph order N.

Bibliography

[1] D. Chandler, *Introduction to Modern Statistical Mechanics*, (Oxford University Press, Oxford 1987).

[2] J. M. Yeomans, *Statistical Mechanics of Phase Transitions*, (Larendon Press, Oxford 1992).

[3] L. E. Reichl, *A Modern Course in Statistical Physics*, (John Wiley & sons, New York 1998).

[4] E. Ising, Z. Phys. **31**, 253 (1925).

[5] L. Onsager, Phys. Rev. **65**, 177 (1944).

[6] M. Mézard, G. Parisi, M. A. Virasoro, *Spin Glasses and Beyond*, (World Scientific, Singapore 1987).

[7] S. F. Edwards and P. W. Anderson, J. Phys. **5**, 965 (1975).

[8] D. Sherrington and S. Kirkpatrick, Phys. Rev. Lett. **35**, 1792 (1975).

[9] G. Parisi, J. Phys. A **13**, 1101 (1980).

[10] M. Talagrand, to app. in Ann. Math. The result was first announced in C.R.A.S. **337**, 111 (2003).

[11] I. Kanter and H. Sompolinsky, Phys. Rev. Lett. **58**, 164 (1987).

[12] Viana and A. Bray, J. Phys. C **18**, 3037 (1985).

[13] C. de Dominicis and P. Mottishaw, J. Phys. A **20**, L1267 (1987).

[14] P. Mottishaw and C. de Dominicis, J. Phys. A **20**, L375 (1987).

[15] K. Y. M. Wong and D. Sherrington, J. Phys. A **21**, L459 (1988).

[16] Y. Goldschmidt and P. Y. Lai, J. Phys. A **23**, L775 (1990).

[17] R. Monasson, J. Phys. A **31**, 513 (1998).

[18] M. Mézard and G. Parisi, Eur. Phys. J. B **20**, 217 (2001).

[19] S. Franz and M. Leone, J. Stat. Phys. **111**, 535 (2003).

[20] H. Bethe, Proc. Roy. Soc. Lond. **150**, 552 (1935).

[21] R. Peierls, Proc. Roy. Soc. Lond. **154**, 207 (1936).

6 Algorithms and numerical results for vertex covers

In this part, we present algorithms for studying the *vertex-cover problem* (VC). After having introduced some basic notation, we first introduce two *linear-time heuristics*. The first allows us to find solutions close to the global optimum, but we cannot give a bound on the quality of the result. The second heuristic is not as efficient, but allows for a calculation of a bound. Then, a *branch-and-bound algorithm* is presented, which calculates exact optima, but *a priori* requires exponential time resources. It is used to derive some exact numerical results which allow us to draw a phase diagram of VC for random graphs from the Erdős–Rényi ensemble $\mathcal{G}(N, c/N)$ for finite average connectivity. We show evidence for the existence of a *sharp phase transition*. Close to this phase transition, the hardest to solve problem instances are found, whereas quite far away, the problem becomes almost surely easy. The branch-and-bound algorithm exhibits an *easy–hard transition*. Next, we present another heuristic, called *leaf removal* which allows for a substantial speedup for random graphs having an average vertex degree $c < e$, where $e \approx 2.718$ is the Euler number. Further on we introduce a *dynamical algorithm*, which is based on viewing the vertex-cover problem as a model for a gas of hard-core particles, and which mimics the physical compaction dynamics in order to approach small VCs. Next, we explain the concept of the *backbone*. We give an efficient algorithm to calculate it, which again is applied to random graphs. At the end, we explain how to analyze the *cluster structure* of the solutions of VC, numerically.

6.1 Definitions

The vertex-cover problem is an NP-hard problem in graph theory, as we have seen before in the complexity-theoretic chapter. As a reminder, we recall the definition and introduce some further concepts which are useful in the description of algorithms.

To do so, we take a dynamical view of the covering process, since any algorithm for determining covers exhibits some kind of dynamic. One can imagine that the algorithm places *covering marks* at the vertices of a graph $G = (V, E)$, one after the other. Let, at a certain time, V' be a set collecting these vertices. We will denote the members of V' *covered*, and all other vertices *uncovered*. Since it is the aim to place covering marks on at least one end-point of each edge, an edge $\{i, j\} \in E$ is analogously called *covered* iff at least one of its end-vertices is *covered*, $i \in V'$ or $j \in V'$. The edge is called uncovered iff both end-points are uncovered. According to the definition, V' thus constitutes a *vertex cover* iff *all* edges are covered. In this case, we call the graph covered as well.

Note that vertex covers always exist, e. g., the full vertex set V is always a vertex cover. They are not unique, because deleting, e. g., for each vertex $i \in V$, the set $V \setminus \{i\}$ is a vertex cover as well.

Example: Vertex cover

Consider the graph shown in the left half of Fig. 6.1. Vertices 1 and 2 are *covered* ($V' = \{1, 2\}$), while the other vertices $3, 4, 5$ and 6 are *uncovered*. Thus, edges $\{1, 3\}$, $\{1, 4\}$ and $\{2, 3\}$ are *covered* while edges $\{3, 4\}$, $\{3, 5\}$, $\{4, 6\}$ and $\{5, 6\}$ are *uncovered*. Hence, the graph is not *covered*.
In the right half of Fig. 6.1 vertices 4 and 5 are also *covered*. Thus, edges $\{3, 4\}$, $\{3, 5\}$, $\{4, 6\}$ and $\{5, 6\}$ are now *covered* as well. This means all edges are *covered*, i. e., the graph is *covered* by $V_{\rm vc} = \{1, 2, 4, 5\}$, thus $V_{\rm vc}$ is a vertex cover.

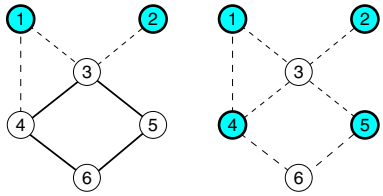


Figure 6.1: Graphs and covers. *Covered* vertices are shown in bold and dark, *covered* edges are indicated by dashed lines. Left: a partially covered graph. Vertex 1 and 2 are *covered*. Thus, edges $\{1, 3\}$, $\{1, 4\}$, and $\{2, 3\}$ are *covered*. Right: If we also cover vertices 4 and 5, the graph becomes covered.

□

The *vertex-cover decision problem* asks whether, for a given graph G, there are VCs $V_{\rm vc}$ of fixed given cardinality $X = |V_{\rm vc}|$, we define $x = X/N$. In other words we are interested if it is possible to cover all edges of G by covering xN suitably chosen vertices, i. e., by distributing xN covering marks. To measure the extent a graph is not covered, we introduce a *cost function* (i. e., a Hamiltonian) mapping arbitrary vertex subsets $V' \subset V$ to the number of uncovered edges,

$$H(V') = |\{\{i, j\} \in E \mid i, j \notin V'\}| \,, \tag{6.1}$$

and the corresponding constraint *ground-state energy*

$$E(G, x) = Ne(G, x) = \min\{H(V') \mid V' \subset V,\ |V'| = xN\} \tag{6.2}$$

as the minimum realizable cost of putting exactly xN covering marks. Thus, a graph G is coverable by xN vertices if $e(G, x) = 0$. This means that you can answer the VC decision problem by first solving a *minimization problem*, and then testing whether or not the minimum $e(G, x)$ is zero.

For the preceding case, the energy was minimized with fixed $X = xN$. The decision problem can also be solved by solving another *optimization problem*. For a given graph G, we look for

the *minimum vertex cover*, i. e., a vertex cover of minimum size

$$X_{\mathrm{c}}(G) = Nx_{\mathrm{c}}(G) = \min\{|V'| \mid H(V') = 0\}\,. \tag{6.3}$$

Thus, here the number of vertices in the subset is minimized, while the energy is kept at zero. The answer to the vertex cover decision problem is "yes" iff $X \geq X_{\mathrm{c}}$.

In the next two sections, numerical methods to solve the vertex-cover problem are presented. Note that we have to distinguish between the two presented ways to state the problem. Either we minimize the energy at given subset cardinality, or we directly construct a minimum VC. We always present the algorithms in a form in which they are suitable for the second kind of problem. Afterwards, we outline how the methods can be changed to treat problems of the first kind.

6.2 Heuristic algorithms

Let us start with two heuristics. We introduce a fast heuristic, which will be utilized also within the exact algorithm discussed in the next section. It can, however, be applied stand-alone as well. In this case only an approximation of the true minimum vertex cover is calculated, which is, empirically, found to differ only by a few percent from the exact value. No exact bounds are available for this method, i. e., there is no rigorous control of its performance. For this reason, we present another approximation algorithm, which allows for a bound with respect to the true optimum. Unfortunately, the bound is not very good. All methods can easily be implemented in C/C++ via the help of the LEDA library [1] or the Boost library [2] which offer many useful data types and algorithms for graph problems.

We begin with a fast greedy heuristic. The basic idea is to cover as many edges as possible by using as few vertices as necessary. Thus, it is favorable to cover vertices with a high degree. This step can be iterated, while the degree of the vertices is adjusted dynamically by removing edges which are *covered*. This leads to the following greedy algorithm, which returns an approximation of the minimum vertex cover V_{vc}, where the size $|V_{\mathrm{vc}}|$ is an upper bound of the true minimum vertex-cover size:

algorithm greedy-cover($G = (V, E)$
begin
  initialize $V_{\mathrm{vc}} = \emptyset$;
  while there are *uncovered* edges (i. e., $E \neq \emptyset$) **do**
  begin
    take one vertex i of highest current degree d_i;
    mark i as *covered*: $V_{\mathrm{vc}} = V_{\mathrm{vc}} \cup \{i\}$;
    remove from E all edges $\{i, j\}$ incident to i;
  end;
  return(V_{vc});
end

Example: Greedy heuristic

To demonstrate how the heuristic operates, we consider the graph shown in Fig. 6.2. The vertices 3,7 and 8 have maximum degree 3. Let us assume that vertex 3 is first covered. Hence, the incident edges $\{1,3\}$, $\{3,4\}$, and $\{3,6\}$ are removed. Still, vertices 7 and 8 have the highest degree 3. We assume that in the second iteration vertex 7 is covered, resulting in deleting edges $\{4,7\}$, $\{6,7\}$ and $\{7,8\}$. In the two final iterations, e. g., vertices 5 and 8 may be covered. Then the algorithm stops, because all edges are *covered*. This cover has size 4, and is indeed a minimum-vertex cover.

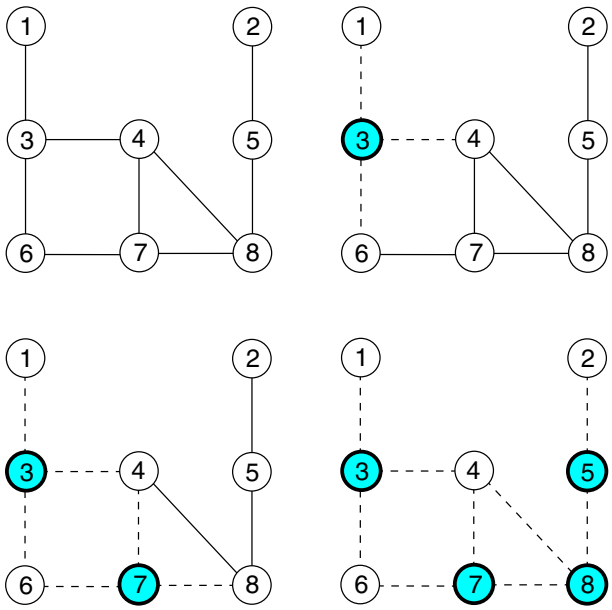


Figure 6.2: Example of the vertex-cover heuristic. Upper left: Initial graph. Upper right: Graph after the first iteration, vertex 3 has been covered (shown in bold) and the incident edges have been removed (shown with dashed line style). Bottom: Graph after second and fourth (final) iteration.

□

In the preceding example we have seen that the heuristic is sometimes able to find a true minimum vertex cover. But this is not always the case. In Fig. 6.3 a simple counterexample is presented, where the heuristic fails to find the true minimum vertex cover. First, the algorithm covers the root vertex, because it has degree 3. Thus, three additional vertices have to be subsequently covered, i. e., the heuristic covers four vertices. But the minimum vertex cover has only size 3, as indicated in Fig. 6.3.

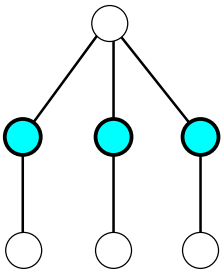


Figure 6.3: A small sample graph with minimum vertex cover of size 3. The vertices belonging to the minimum $V_{\rm vc}$ are indicated by dark/bold circles. For this graph, the greedy heuristic fails to find the true minimum cover, because it starts by covering the root vertex, which owns the highest degree 3.

The heuristic can easily be altered for the case in which the number X of *covered* vertices is fixed and we ask for a minimum number of uncovered edges. Then the iteration has to stop as well, when the size of the cover set has reached X. In case a vertex cover is found before X vertices are covered, arbitrary vertices can be added to the vertex-cover set $V_{\rm vc}$ until $|V_{\rm vc}| = X$.

Unfortunately, it has not been possible so far to derive a rigorous bound on how the result of the heuristic compares to the true minimum vertex cover. Deriving such a bound is possible for the following algorithm [3], although the bound is not very good. The algorithm is based on the relationship between vertex covers and matchings. We recall from Chap. 3, that a matching is a subset of edges, such that each vertex is contained at most once in each matching. The following algorithm stores the vertex cover being built in $V_{\rm vc}$ and the matching in M.

algorithm 2-approximation($G = (V, E)$
begin
  initialize $V_{\rm vc} = \emptyset$;
  initialize $M = \emptyset$;
  while there are *uncovered* edges (i. e., $E \neq \emptyset$) **do**
  begin
    take one arbitrary edge $\{i, j\} \in E$;
    mark i and j as *covered*: $V_{\rm vc} = V_{\rm vc} \cup \{i, j\}$;
    add $\{i, j\}$ to the matching: $M = M \cup \{\{i, j\}\}$;
    remove from E all edges incident to i or j;
  end;
  return($V_{\rm vc}$);
end

Example: 2-Approximation heuristic

To demonstrate, how the 2-approximation heuristic operates, we consider again the graph from the previous example, see Fig. 6.4. We assume that the algorithm first takes edge $\{3,4\}$, hence after the first iteration $V_{\mathrm{vc}} = \{3,4\}$, $M = \{\{3,4\}\}$ and the edges $\{1,3\}$, $\{3,4\}$, $\{3,6\}$ and $\{4,7\}$ are covered and removed from the graph. In the next iteration edge $\{7,8\}$ may be chosen, hence we have now $V_{\mathrm{vc}} = \{3,4,7,8\}$, $M = \{\{3,4\},\{7,8\}\}$ and edges $\{4,8\}$, $\{5,8\}$, $\{6,7\}$ and $\{7,8\}$ are covered and removed from E. Now only edge $\{2,5\}$ is left. Hence, after the final iteration, we have $V_{\mathrm{vc}} = \{2,3,4,5,7,8\}$ and $M = \{\{3,4\},\{7,8\},\{2,5\}\}$.

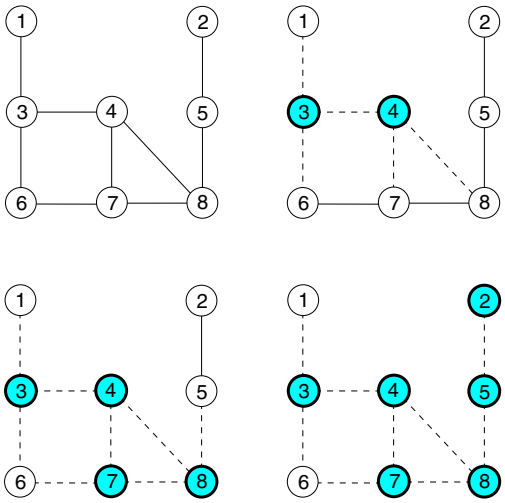


Figure 6.4: Example of the 2-approximation heuristic. Upper left: initial graph. Upper right: graph after the first iteration, vertices 3 and 4 have been covered (shown in bold) and the incident edges removed (shown with dashed line style). Bottom: graph after second and final iteration.

Note that, in the case when the algorithm "chose" the edges, e. g., in the order $\{1,3\}$, $\{2,5\}$, $\{6,7\}$ and $\{4,8\}$, then the vertex cover would contain all eight vertices, i. e., twice the size of the minimum vertex cover. We will show below that the algorithm never achieves a worse result.

On the other hand, the 2-approximation algorithm will never be able to "find" the minimum cover for this graph, because in the minimum cover, e. g., $V_{\mathrm{vc}}^{\min} = \{3,5,7,8\}$, there are always vertices which have no neighbor in $V_{\mathrm{vc}}^{\min}$. This is not possible within the 2-approximation algorithm, because pairs of neighbors are always added to V_{vc}. □

Each edge is touched exactly once by the 2-approximation algorithm, hence the running time is of order $\mathcal{O}(|E|)$, if one assumes a constant execution time for all fundamental operations. Furthermore, the algorithm removes within the loop only *covered* edges from E. Since the algorithm halts when E is empty, all edges are covered, hence $V_{\rm vc}$ is a vertex cover.

The size $|V_{\rm vc}|$ of the vertex cover is, at most, twice the size of the minimum vertex cover $V_{\rm vc}^{\rm min}$: $|V_{\rm vc}| \leq 2|V_{\rm vc}^{\rm min}|$.

Proof:

The algorithm also constructs a matching M. Since the algorithm adds two vertices to $V_{\rm vc}$ for each edge which is added to M, we have exactly

$$|V_{\rm vc}| = 2|M| \,. \tag{6.4}$$

Since no vertex appears twice in the edges of the matching by definition, i. e., the edges do not "touch" each other, one has to cover at least one vertex per edge of any matching, hence also per edge of M. This means for the minimum vertex cover $V_{\rm vc}^{\rm min}$ we have

$$|V_{\rm vc}^{\rm min}| \geq |M| \,. \tag{6.5}$$

Combining Eqs (6.4) and (6.5) we get $|V_{\rm vc}| = 2|M| \leq 2|V_{\rm vc}^{\rm min}|$. QED

6.3 Branch-and-bound algorithm

So far we have presented two simple heuristics to find approximations of minimum vertex covers. Next, an exact algorithm is explained: *branch-and-bound*, which incorporates the first heuristic to gain high efficiency. Without the heuristic, the algorithm would still be exact, but it would run considerably slower.

The basic idea of the method is as follows. Again we are interested in a VC of minimum size. As each vertex becomes either *covered* or *uncovered*, there are 2^N possible configurations which can be arranged as leaves of a binary configuration tree, see Fig. 6.5. At each node, the two subtrees represent the subproblems where the corresponding vertex is either *covered* ("left subtree") or *uncovered* ("right subtree"). Vertices which have not been touched at a certain level of the tree, are said to be *free*. Note that for different nodes on the same level of the tree, the vertices corresponding to the node do not have to be the same in different subtrees, e. g., n_2 may be different from n_2'. This depends on the heuristic which is used to determine the current vertex at each node of the configuration tree. The algorithm does not have to descend further into the tree when a cover has been found, i. e., when all edges are *covered*. Then the search continues in higher levels of the tree for a cover which has possibly a smaller size, i. e., *backtracking* occurs. Since the number of nodes in a tree grows exponentially with system size, algorithms which are based on configuration trees have a running time which may grow exponentially with the system size. This is not surprising, since the minimum-VC problem is NP-hard, so all known exact methods exhibit an exponential growing worst-case time complexity.

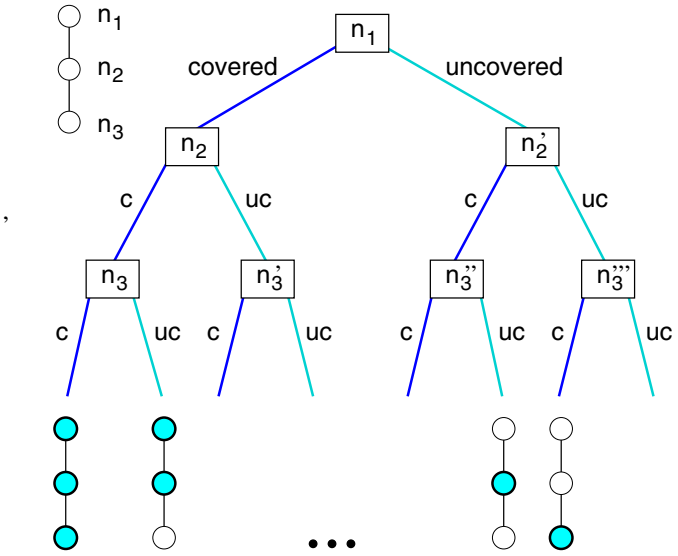


Figure 6.5: Binary configuration tree for the VC. Each node of the configuration tree corresponds to a vertex which is either *covered* ("left subtree") or *uncovered* ("right subtree").

To decrease the running time, the algorithm presented below makes use of the fact that only proper vertex covers are to be obtained. Therefore, when a vertex i is marked as *uncovered*, all neighboring vertices can be *covered* immediately. For these vertices, only the left subtrees are present in the configuration tree, hence the size of the tree is already reduced.

A further substantial speedup can be obtained by applying the branch-and-bound approach [4, 5]. The idea is that some subtrees of the configuration tree are omitted, i. e., they are not visited at all, by introducing a *bound*. This is achieved by storing three quantities, assuming that the algorithm is currently found at some node in the configuration tree:

- The *best* size of the smallest vertex cover found in subtrees visited so far (initially $best = N$).
- X denotes the number of vertices which have been covered so far (in higher levels of the tree).
- A table of *free* vertices indexed by the *current* degree d_i is always kept, i. e., for each vertex, the number of currently *uncovered* incident edges.

Thus, to achieve a better solution than *best*, at most $F = best - X$ vertices are allowed to additionally be covered in a subtree of the current node. The number of edges coverable in this way is obviously bounded from above by the sum $D = \sum_{l=1}^{F} d_l$ of the F highest current degrees. If this number is smaller than the total number of not yet covered edges, we know that the subtree cannot contain a smaller VC of the full graphs. It can be omitted for sure. Note that some edges may exist between the F vertices of the highest current degree, they are

counted twice in D. Therefore the bound is not necessarily tight, a subtree may be entered, even if it contains no smaller VC. Other types of bounds are mentioned in Ref. [5], e. g., the size of a maximal matching is a lower bound for a minimum vertex cover.

The algorithm can be summarized as below. The size of the smallest cover is stored in *best*, which is passed by reference (i. e., the variable, not its value is passed). The current number of *covered* vertices is stored in variable X:

algorithm branch-and-bound(G, *best*, X)
begin
  if all edges are *covered* **then**
  begin
    if $X <$ *best* **then** $best := X$
    return;
  end;
  calculate $F = best - X$; $D = \sum_{l=1}^{F} d_l$;
  if $D <$ number of *uncovered* edges **then**
    return; **comment** bound;
  take one *free* vertex i with the largest current degree d_i;
  mark i as *covered*;**comment** left subtree
  $X := X + 1$;
  remove from E all edges $\{i, j\}$ incident to i;
  branch-and-bound(G, *best*, X);
  reinsert all edges $\{i, j\}$ which have been removed;
  $X := X - 1$;
  if ($F >$ number of current neighbors) **then**
  begin **comment** right subtree;
    mark i as *uncovered*;
    for all neighbors j of i **do**
    begin
      mark j as *covered*; $X := X + 1$;
      remove from E all edges $\{j, k\}$ incident to j;
    end;
    branch-and-bound(G, *best*, X);
    for all neighbors j of i **do**
      mark j as *free*; $X := X - 1$;
    reinsert all edges $\{j, k\}$ which have been removed;
  end;
  mark i as *free*;
  return;
end

Example: Branch-and-bound algorithm

Here we consider again the graph from Fig. 6.2. During the first descent into the configuration tree, the branch-and-bound algorithm operates exactly as the heuristics. Iteratively vertices of highest current degree are taken, covered, and the incident edges removed. The recursion stops the first time the graph is covered. This situation is shown in Fig. 6.6, where the graph and the corresponding current configuration tree are displayed. Since $X = 4$ vertices have been covered, $best := 4$.

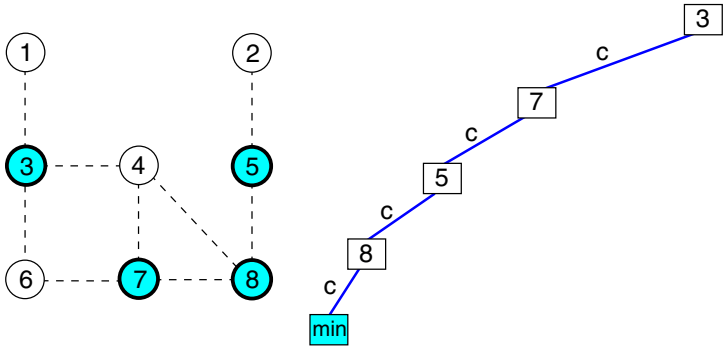


Figure 6.6: Example of the branch-and-bound algorithm. Result after the first full cover has been found. Left: graph. Right: configuration tree. In the graph, covered vertices are shown by bold and dark circles, covered edges indicated by dashed lines. The current node of the configuration tree is highlighted as well, c=*cover*, uc=*uncover*.

Then the algorithm returns to the preceding level of the configuration tree. Vertex 8 is now set *uncovered*. All its *uncovered* neighbours are *covered*, i. e., vertex 4 in this case. The resulting situation is shown in Fig. 6.7. At the next level of the configuration tree, it is detected that again a full vertex cover has been found, but not a smaller one. Hence, the algorithm backtracks to the previous level.

Both possibilities for vertex 8 have been considered, hence vertex 8 is set to *free* again and the algorithm backtracks another level. Now the second possibility for vertex 5 is considered, i. e., it is *uncovered*, while its neighbours, vertices 2 and 8 are *covered*, see Fig. 6.8. Again a vertex cover of size 4 has been found, no further descent into the tree beyond the next level is necessary.

This means that the treatment of vertex 5 is finished, it is *freed* again, and the algorithm backtracks one level, to continue considering vertex 7. It is *uncovered*, and its neighbours, vertices $4, 6$ and 8 are *covered*, see Fig. 6.9. In the subsequent recursive call, it is found that no cover has been found, because edge $\{2, 5\}$ is uncovered. Hence, the bound is evalated. Since the current number of *covered* vertices is $X = 4$, we obtain $F = best - X = 0$. This results trivially in $D = 0$, which is smaller than the number of *uncovered* edges. Therefore, the bound becomes ac-

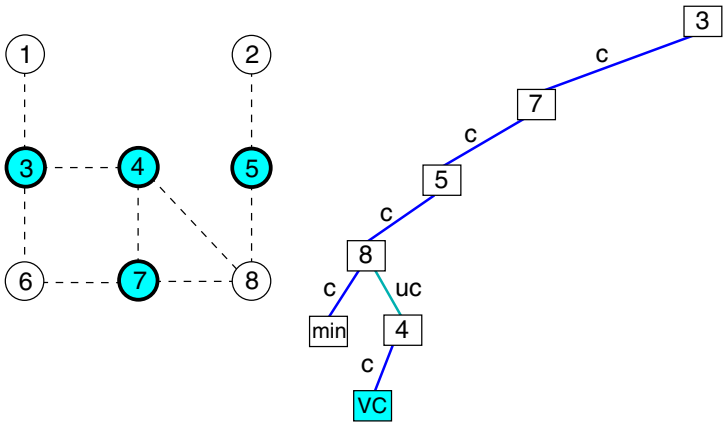


Figure 6.7: Example of the branch-and-bound algorithm: Situation after vertex 8 has been *uncovered* and its neighbour 4 has been *covered*. A new vertex cover has been found, but not a smaller than before, indicated by a "VC" in the current node.

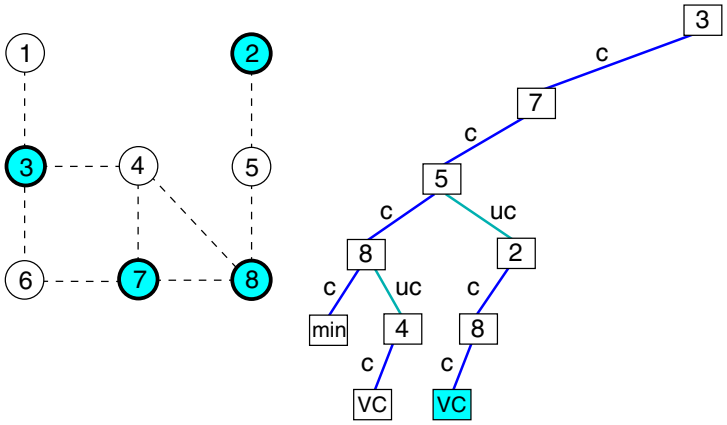


Figure 6.8: Example of the branch-and-bound algorithm: Situation after vertex 5 has been *uncovered* and its neighbours 2,8 have been *covered*.

tive, the left and right subtrees of the current node are omitted, and the algorithm backtracks to the previous level of the configuration tree.

Thus, the treatment of vertex 7 is finished and the algorithm backtracks further to the top level and *uncovers* vertex 3. Thus, its neighbours, vertices 1,4 and 6 are *covered*, see Fig. 6.10. Again no cover has been obtained, hence the bound is evaluated during the next recursive call to the algorithm. Now we have $X = 3$ vertices *covered*, hence we can cover $F = best - X = 4 - 3 = 1$ additional vertices, i. e., one. Vertex 8 has the highest current degree $d_8 = 2$, hence $D = 2$. But the number of uncovered edges is 3. Thus, the bound becomes active in a non-trivial way, the algorithm returnes to the top level and terminates.

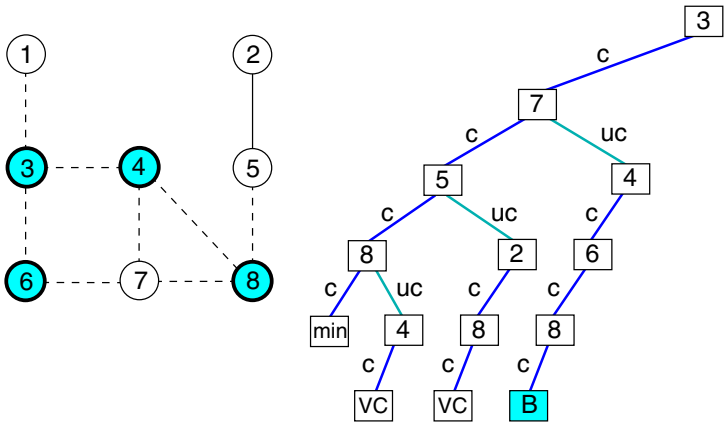


Figure 6.9: Example of the branch-and-bound algorithm: Situation after vertex 7 has been *uncovered* and its neighbours 4,6,8 have been *covered*. Here the bound is active, indicated by a "B" in the current node of the configuration tree.

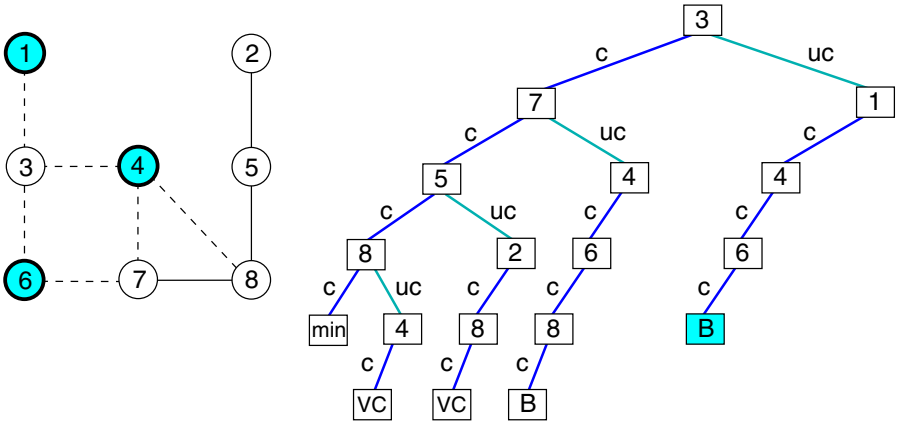


Figure 6.10: Example of the branch-and-bound algorithm. Siutation after vertex 3 has been *uncovered* and all its neighbours *covered*. Again the bound becomes active.

Hence, the minimum vertex cover has indeed size $X = 4$, as found by the heuristics. Note that the configuration tree contains only 18 nodes, compared to 511 nodes (with $2^8 = 256$ leaves) of the complete configuration tree. □

For every calculation of the bound, one has to access the F vertices of largest current degree. Therefore, it is favorable to implement the table of vertices as two arrays v_1, v_2 of sets of vertices. The arrays are indexed by the current degrees of the vertices. The sets in the first array v_1 contain the F *free* vertices of the largest current degree, while the other array contains

all other *free* vertices. Every time a vertex changes its degree, it is moved to another set, and eventually even changes the array. Also, in case the mark (*free/covered/uncovered*) of a vertex changes it may be entered in or removed from an array and possibly the smallest degree vertex of v_1 is moved to v_2 or vice versa. Since we are treating graphs of finite average connectivity, this implementation ensures that the running time spent in each node of the graph is growing slower than linearly in the graph size[1]. For the sake of clarity, we have omitted the update operations for both arrays from the algorithmic presentation.

The algorithm, as it has been presented, is suitable for solving the optimization problem, that is, finding the smallest feasible size $X_{\mathrm{c}} = Nx_{\mathrm{c}}$ of a vertex cover, i. e., the minimum number of *covered* vertices needed to cover the graph fully. The algorithm can be easily modified to treat the problem, where the size $\tilde{X} = |V_{\mathrm{vc}}|$ is given and a configuration with minimum energy is to be found, i. e., the case where the graph may not be fully coverable. Then, in *best*, not the current smallest size of a vertex cover but the smallest number of uncovered edges (i. e., the energy) is stored. If X again denotes the current number of covered vertices at any node in the configuration tree, the algorithm can cover $F = \tilde{X} - X$ additional vertices. Again $D = \sum_{l=1}^{F} d_l$ is the sum of the F highest degrees. A subtree should not be entered, if the number of uncovered edges so far minus the maximum possible number of edges coverable within the subtree is larger than the current optimum. Hence, the bound becomes active, if $D+$*best* is smaller than or equal to the current number of uncovered edges. Furthermore, when a vertex is *uncovered*, the step where all neighbors are *covered* cannot be applied, because the configuration of the lowest energy may not be a VC. On the other hand, if a VC has been found, i. e., all edges are covered, the algorithm can stop, because no better solution can definitely be obtained. But the algorithm can stop only for the case when *one* optimum has to be obtained. If all minima have to be enumerated, the algorithm has to proceed and the bound becomes active only when $D +$ *best* is strictly smaller (not equal) to the current number of *uncovered* edges.

6.4 Results: Covering random graphs

The algorithm has obviously an exponential worst-case behavior since the configuration tree is *a priori* exponentially large. Following the general strategy of going from worst to *typical cases* explained in Sec. 4.9, we apply the aforementioned algorithmic methods to randomized problem samples. A natural choice is provided by the Erdős–Rényi ensemble $\mathcal{G}(N, c/N)$ of random graphs of finite average degree, which was discussed in some detail in Sec. 3.3.

Although the branch-and-bound algorithm is very simple, we can treat random graphs up to size $N = 140$ if we restrict the average degrees to the region $c < 10$. It is, however, difficult to compare our algorithm with more elaborate ones existing in the literature [5, 6], because they have usually been tested on a different graph ensemble in which edges appear with a

[1] Efficient implementation of *sets* requires at most $\mathcal{O}(\log S)$ time for the operations, where S is the size of a set. In this case also a double indexed structure is possible allowing all operations to be performed in constant time. Double indexed structure means that sets are also implemented as arrays and, for each vertex, the current position in the corresponding array must be stored.

fixed probability, independently of the graph size (high-connectivity regime). Nevertheless, in the computer-science literature, usually graphs with up to 200 vertices are treated, which is slightly larger than the systems considered here. On the other hand, the algorithm presented here has the advantage that it is easy to implement and its power is sufficient to study interesting problems.

First, we consider the problem of minimizing the energy (6.2) as a function of the relative VC size x. The average of the energy density over the random graph ensemble $\mathcal{G}(N, c/N)$ for fixed average degree c will be denoted by $e(c, x) = \overline{e(G, x)}$. Another interesting quantity in this context is the probability $P_{\mathrm{cov}}(c, x)$ that a $\mathcal{G}(N, c/N)$-graph is completely coverable with xN covering marks. Two limiting cases are obvious. For $x = 0$, we have no covering marks, the energy density becomes $e(c, 0) = c$, the graphs are almost surely uncoverable, $P_{\mathrm{cov}}(c, 0) \to 0$ (only the extremely improbable case of a graph without edges would be covered). For $x = 1$, we can cover all vertices and consequently all edges, $e(c, 1) = 0$ and $P_{\mathrm{cov}}(c, 0) = 1$. It is also clear that this energy density, seen as a function of x at fixed c is monotonously decreasing, because a higher number of covering marks allow us to cover a higher number of edges. Using the same argument, we conclude that the probability P_{cov} is monotonously increasing.

In Fig. 6.11 these two quantities are plotted for average degree $c = 2.0$, analogous results are found for other c-values. Curves are shown for different system sizes $N = 25, 50, 100$, and data are averages over 10^3 ($N = 100$) to 10^4 ($N = 25, 50$) randomly generated graphs. As expected, the value of $P_{\mathrm{cov}}(2, x)$ increases with the fraction of *covered* vertices. The astonishing result is, however, that the change from values close to zero to those close to one happens in a very restricted x-interval. With growing graph order, this change becomes even steeper. All different curves intersect almost at the same point. This suggests that in the limit $N \to \infty$, in which we are interested, a *sharp threshold* $x_{\mathrm{c}}(c = 2) \approx 0.39$ exists. Above $x_{\mathrm{c}}(c)$ a graph is almost surely coverable, below it is almost surely uncoverable. Thus, in the language of physics, a *phase transition* from an uncoverable phase to a coverable phase occurs. Note that the value $x_{\mathrm{c}}(c)$ of the critical threshold depends on the average connectivity c. The result for the phase boundary $x_{\mathrm{c}}(c)$ as a function of c can be extracted from the simulations and is shown later on.

In Fig. 6.12 the median (i. e., typical) running time of the branch-and-bound algorithm is shown as a function of the fraction x of covered vertices. The running time is measured in terms of the number of nodes which are visited in the configuration tree. Again graphs with $c = 2.0$ were considered and an average over the same realizations has been performed. The most characteristic result is a sharp peak observed in the direct vicinity of the transition point x_{c}. This means that, near the phase transition, those instances are located, which are typically the hardest to solve. Note that for values $x < x_{\mathrm{c}}$, the running time still increases exponentially, as can been seen from the inset of Fig. 6.12. A second important result is that, for values x considerably larger than the critical value x_{c}, the typical running time becomes *linear*. The reason is that the heuristic is already able to find a VC, i. e., the algorithm terminates after the first descent into the configuration tree[2]. So, even if the problem is NP-hard, there are

[2]The algorithm terminates after a full vertex cover of the graph has been found.

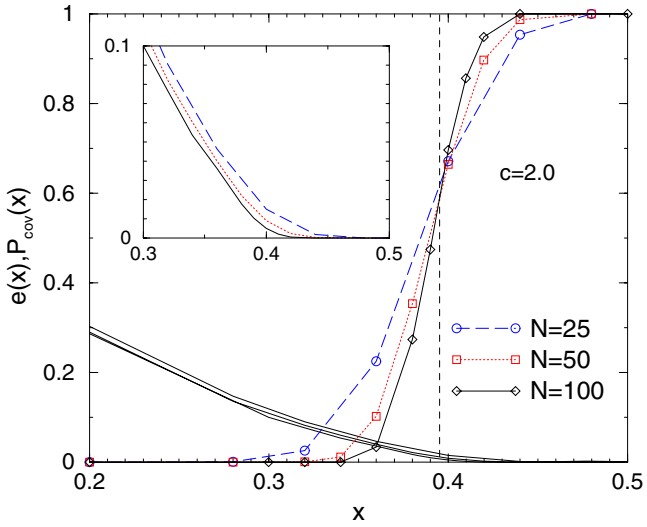


Figure 6.11: Probability $P_{\mathrm{cov}}(2, x)$ that a VC exists for a random graph ($c = 2$) as a function of the fraction x of *covered* vertices. The result is shown for three different system sizes $N = 25, 50, 100$ (averaged over $10^3 - 10^4$ random graphs). Lines are guides to the eyes only. In the left part, where the P_{cov} is zero, the average energy density $e(2, x)$ (see text) is displayed. The inset enlarges the result for the energy in the region $0.3 \leq x \leq 0.5$.

parameter regions where the problem is *typically easy*. This phenomenon will be discussed analytically in great detail in Chap. 8.

Note that phase transitions in a physical system are usually indicated by a divergence of measurable quantities such as specific heat or magnetic susceptibilities. The peak appearing in the time complexity serves as a similar indicator, but is not really equivalent, because the time complexity diverges everywhere, only the rate of divergence is much stronger near the phase transition.

An indicator, which is more closely related to physical quantities, is the *correlation volume*. We define it in the following way. Usually, the minimum energy configuration with given size X is not unique, physically speaking the configuration is *degenerate*, see also Sec. 6.7. If we force one vertex to change its state, other vertices have to change as well to obtain again a minimum energy configuration of size X. The correlation volume is the minimum number of vertices which have to be changed, averaged over the different vertices, which are not frozen to one state in all minimum-energy configurations. As shown in the inset of Fig. 6.13, close to the phase transition, a divergence of the correlation volume can be observed [7]. This is comparable to the divergence of the correlation length at second-order phase transitions. Unfortunately, the calculation of the correlation volume requires the enumeration of all ground states, hence only small system sizes can be treated.

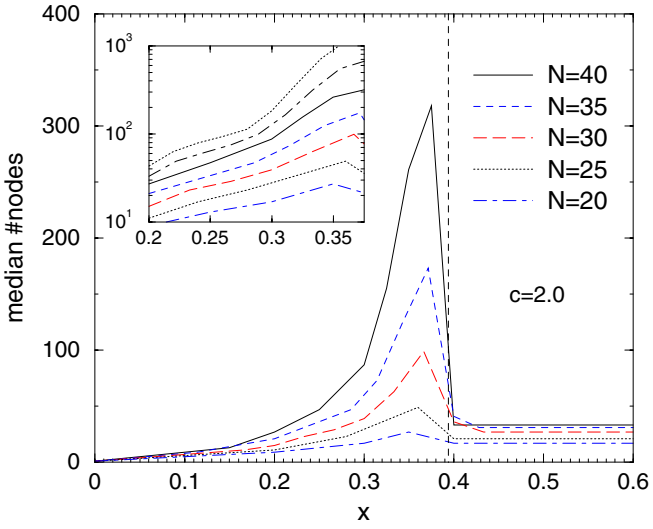


Figure 6.12: Time complexity of the vertex-cover algorithm. We display the median number of nodes visited in the configuration tree as a function of the fraction x of *covered* vertices for graph sizes $N = 20, 25, 30, 35, 40$, the average vertex degree is fixed to $c = 2.0$. The inset shows the region below the threshold on a logarithmic scale, including also data for $N = 45, 50$. The equidistant position of the curves in this representation illustrates that the time complexity grows exponentially with N.

Coming back to the running time, for small values of x in the uncoverable region, the running time is also faster than that close to the phase transition, but still exponential. This is due to the fact that a configuration with a minimum number of *uncovered* edges has to be obtained. If only the question whether a VC exists or not is to be answered, the algorithm can be easily improved[3], such that for small values of x again a polynomial running time will be obtained, cf. Fig. 8.4 in Sec. 8.

Now we are going to numerically calculate the phase diagram, i. e., the position of the critical fraction x_c of covered vertices as a function of c. In this case it is sufficient to calculate for each graph the size $X_c = Nx_c$ of a minimum vertex cover, as done by the second version of the branch-and-bound algorithm presented in the last chapter. To compare with the analytical results which will be presented in the next chapter, one has to perform the thermodynamic limit $N \to \infty$ numerically. This can be achieved by calculating an average value $x_c(N)$ for different graph sizes N. Then one fits a function

$$x_c(N) = x_c + aN^{-b} \tag{6.6}$$

to the data. The form of the function is purely heuristic, no exact justification is known so far for the case of VC. Analogous scaling forms are, however, well known in the *finite-size*

[3] Set $best := 0$ initially.

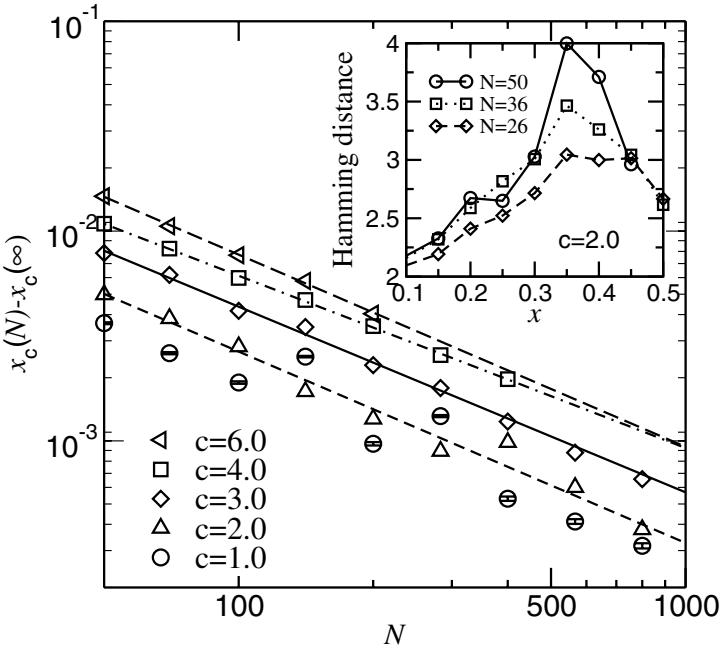


Figure 6.13: Finite-size scaling behavior of the critical cover size. The location of the transition point $x_c(N)$ as a function of graph size N for different average degree c. Inset: scaling of the correlation volume as a function of x for different sizes. Error bars are, at most, of the order of the symbol size.

scaling analysis of second-order phase transitions [8–10], where the location of the transition points in finite systems is governed by $b = 1/\nu$, ν being the critical exponent of the correlation length. As can be seen from Fig. 6.13, the fits match well, although for small average degrees a small but significant scattering of the data occurs. Interestingly, the exponent b describing the finite-size scaling term does not depend much on the connectivity c, for $c > 1$ one obtains $b(c = 2) = 0.91(9)$, $b(3) = 0.88(4)$, $b(4) = 0.82(4)$, and $b(6) = 0.92(11)$. Hence, within error bars, this exponent seems to be universal.

This procedure has been performed for several values of c. The result is indicated in Fig. 6.14 by the symbols. Using probabilistic tools, rigorous lower and upper bounds for this threshold [11] and the asymptotic behavior for large connectivities [12] have been deduced, cf. also the next two sections. These bounds are indicated by dotted and dashed lines in in Fig. 6.14.

In the next chapter, we will show how the problem can be investigated using a statistical physics approach [13]. Let us for a moment anticipate the main result: Up to an average degree $c = e \approx 2.718$ the transition line

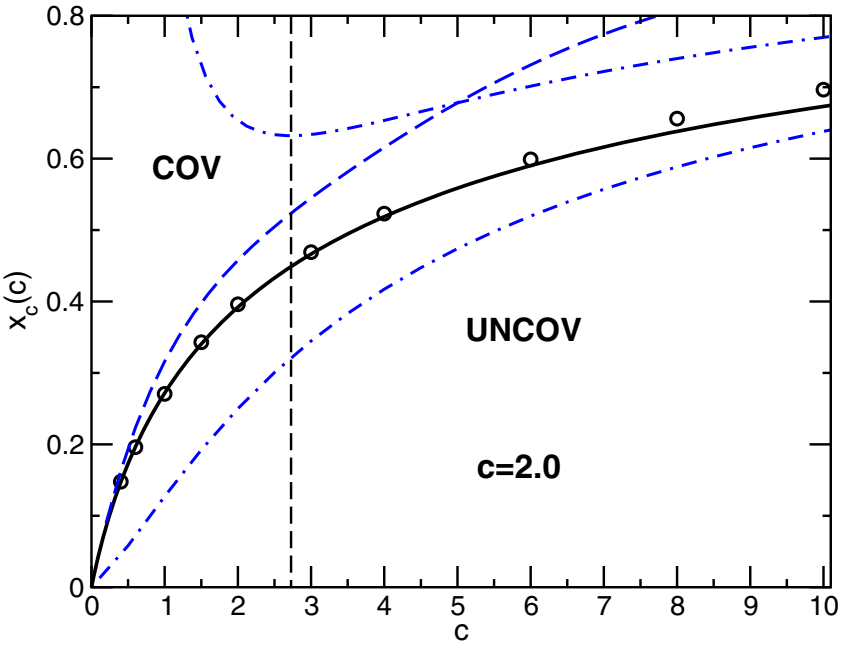


Figure 6.14: Phase diagram showing the fraction $x_c(c)$ of vertices in a minimum vertex cover as a function of the average degree c. For $x > x_c(c)$, almost all graphs have covers with xN vertices, while they have almost surely no cover for $x < x_c(c)$. The solid line shows the analytical result. The circles represent the results of the numerical simulations. Error bars are much smaller than symbol sizes. The upper bound of Harant is given by the dashed line, the bounds of Gazmuri by the dash-dotted lines. The vertical line is at $c = e \approx 2.718$.

is given exactly by

$$x_c(c) = 1 - \frac{2W(c) + W(c)^2}{2c}, \tag{6.7}$$

where $W(c)$ is the Lambert-W function defined by $W(c)\exp(W(c)) = c$. The transition is shown in the phase diagram in Fig. 6.14 by a solid line. As already conjectured on the basis of numerical data, for $x > x_c(c)$, a random graph is almost surely coverable, for $x < x_c(c)$ the available covering marks are not sufficient. For higher connectivities no exact result for $x_c(c)$ is known. Note, however, that the region where the exact result has been obtained, extends fairly well into the percolating regime of random graphs, since the percolation threshold is $c_{\mathrm{crit}} = 1.0 < 2.718$, cf. Sec. 3.3.

As is to be expected, the data obtained from simulations followed by a finite-size scaling analysis, and the analytical result, agree very well in the region $c < e \approx 2.718$. For larger degrees, where the analytical result is not exact, slowly but systematically growing deviations show up. A stronger deviation between numerical and analytical results can be observed when studying the *backbone*, a quantity which is defined in Sec. 6.7 of this chapter.

6.5 The leaf-removal algorithm

To obtain the phase diagram of vertex covers of random graphs in the preceding section, vertex covers of minimum size were calculated. Now we present an extension of the previously introduced algorithm, which works in the range of connectivities $c < e$ in a much better way than the pure branch-and-bound approach. The algorithm, so-called *leaf removal*, exhibits typically a polynomial running time in this region. It is suited for the case, where vertex covers of minimum size are to be obtained.

The basic idea of the algorithm is as follows. Since only full vertex covers are to be considered, *all* edges must be covered by the algorithm. Now we consider the *leaves* of the given graph, i. e., the vertices i having degree 1. Clearly, the edge $\{i, j\}$ connecting i to its neighbor j also has to be covered. This means that, for any proper VC $V_{\rm vc}$, at least one of i and j has to be covered ($i \in V_{\rm vc}$ or $j \in V_{\rm vc}$). When $i \in V_{\rm vc}$ only the edge $\{i, j\}$ is covered, and no other edge. Hence, if we are looking for just one minimum VC, we cannot make a mistake by covering only vertex j. Then edge $\{i, j\}$ is covered and maybe edges to other neighbors of j as well. After covering j, all edges adjacent to j are covered and can effectively be removed from the graph. This removal decreases the degree of the further neighbors of vertex j, so some of them may become new leaves in the reduced graph. This means, that not only are all leaves and their neighbors in the given graph considered, but other vertices may be affected after some iteration steps, see the example below.[4] The algorithm can be summarized as follows:

algorithm leaf-removal ($G = (V, E)$)
begin
  Initialize $V' = \emptyset$
  while there are leaves i (i. e.vertices with degree $d_i = 1$) **do**
  begin
    Let j be the neighbor of a leaf i
    cover j, i. e., $V' = V' \cup \{j\}$
    Remove all edges adjacent to j from E
    Remove i and j from V
  end
  return (V')
end

The algorithm obviously has a linear running time in the number of vertices. Once the reduced graph has no more leaves, the algorithm stops. In principle, it is possible that this directly leads to a VC of the full graph, e. g., for a tree or in the example below. The remaining non-trivial part of the graph, or more precisely the subgraph induced by its non-isolated vertices, is called the *core*. Since all vertices of a core have a degree of at least two, the core is similar to the 2-core introduced in Chap. 3. The difference is that here the leaves *and* their neighbors are removed iteratively, while to obtain the 2-core, only the leaves are removed iteratively.

[4] A similar situation was considered in the q-core algorithm in Chap. 3.

Example: Leaf removal

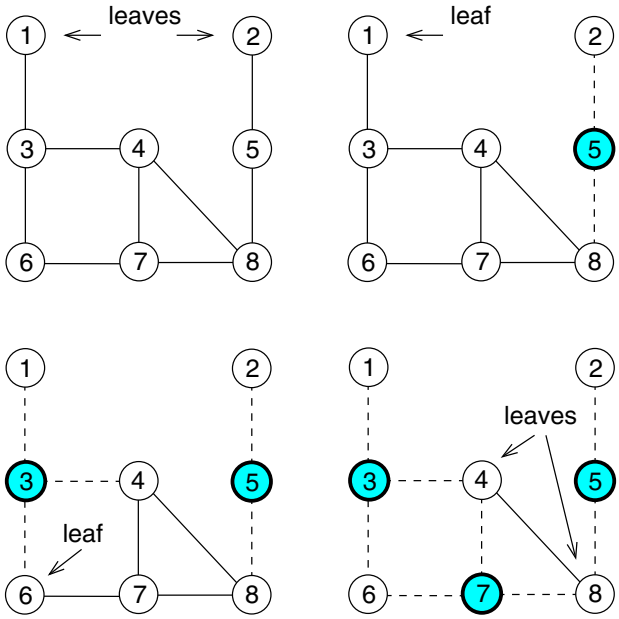


Figure 6.15: Example of the leaf-removal algorithm. Upper left: initial graph, vertices 1 and 2 are leaves. Upper right: graph after the first iteration, vertex 5 has been covered (shown in bold) and the incident edges removed (shown with dashed line style). Bottom: graph after second and third iteration.

To demonstrate, how the algorithm operates, we consider again the graph shown in Fig. 6.2. This graph has two leaves, vertices 1 and 2, as indicated in the upper left part of Fig. 6.15. In the first iteration, say vertex 5, i. e., the neighbor of the leaf vertex 2 is covered. Hence the edges $\{2, 5\}$ and $\{5, 8\}$ are covered and removed from the set of edges. Vertices 2 and 5 are also removed. Now, the only remaining leaf is vertex 1. Hence, in the next iteration, its neighbor, vertex 3, is covered. By this step, due to the removal of edges $\{1, 3\}$ and $\{3, 6\}$, a new leaf is generated, i. e., vertex 6. This means that in the following iteration, vertex 7 is covered. This results in only one uncovered edge left, i. e., two new leaves are generated, vertices 4 and 8. By covering one of them, no edges remain, in particular no leaves. Leaf removal has, for this specific case, produced a VC which is therefore known to be minimum. □

To find a complete vertex cover, the core has to be treated with a complete algorithm, e. g., the branch-and-bound algorithm introduced in Sec. 6.3. Interestingly, this combinations exhibits

a typically polynomially growing running time for the random-graph ensemble for connectivities $c < e$. The reason is that for $c < e$ the core does not percolate, as shown by Bauer and Golinelli [14]. This means that for such small connectivities the core consists only of connected components each having of order $\log(N)$ vertices, $N = |V|$ being the number of vertices in the initial graph. Each of the connected components can be treated independently with the branch-and-bound algorithm. For each component, the branch-and-bound algorithm exhibits at most an exponential running time in the size of the component, i. e., a polynomial running time in the size N of the graph. Since the leaf-removal part is just $\mathcal{O}(N)$, one has in total an algorithm which takes typically a polynomial running time for $c < e$. For larger connectivities, the core percolates, i. e., it has a size of order N. Hence, obtaining the minimum vertex cover will take typically an exponential number of steps in this case.

The leaf-removal algorithm is a special variant of a more general algorithm presented by Tarjan and Trojanowski [15]. It treats the vertices i of a graph depending on their degree d_i. It is assumed that the graph consists of one connected component, otherwise the algorithm can be applied to all connected components independently. Let $d_{\min}$ be the minimum degree in the graph. For $d_{\min} = 1$, i. e., for leaves, leaf removal is applied, as shown above. Apart from this, the algorithm of Tarjan and Trojanowski contains explicit treatment for all degrees $d_{\min} \leq 5$, for higher minimum degrees the algorithm just branches, i. e., considers both cases when a vertex i is *covered* and when i is not *covered*. Since the algorithmic presentation of the full algorithm takes five pages, we just state as an example the case $d_{\min} = 2$, to give a flavor of the general idea. Then two cases are possible. Either all vertices have degree two, or at least one vertex has degree three.

- In the first case, it follows immediately that all vertices from V form a loop, i. e., one has to cover every second vertex of the loop, see Fig. 6.16a. The size of the minimum vertex cover is in this case $\lceil |V|/2 \rceil$.

- In the second case, there exists at least one vertex i with degree $d_i = 2$ which has a neighbor j_1 with degree $d_{j_1} \geq 3$. Let j_2 be the other neighbor of i. Now there are two subcases

  - The edge $\{j_1, j_2\}$ exists. In this case u, j_1, j_2 from a triangle, see Fig. 6.16b. Similar to the case $d_{\min} = 1$, it is now favorable to cover j_1 and j_2 and remove i, j_1, j_2 and all edges incident to them from the graph.

  - The edge $\{j_1, j_2\}$ does not exist, see Fig. 6.16c. In this case one cannot say anything in advance. One has to treat the cases that either i is *covered* or that j_1 and j_2 are *covered* (i *uncovered*) independently. Hence, the algorithm branches and takes the minimum from the two obtained solutions.

The algorithm of Tarjan and Trojanowski has a worst-case running time $\mathcal{O}(2^{n/3})$. There are other complete algorithms used to find minimum vertex covers, often formulated for equivalent versions of VC, i. e., the maximum independent set and the maximum clique problem. A recent complete algorithm [6], which is simpler to implement than the algorithm of Tarjan and Trojanowski, exhibits a worst-case running time $\mathcal{O}(2^{n/2.63})$.

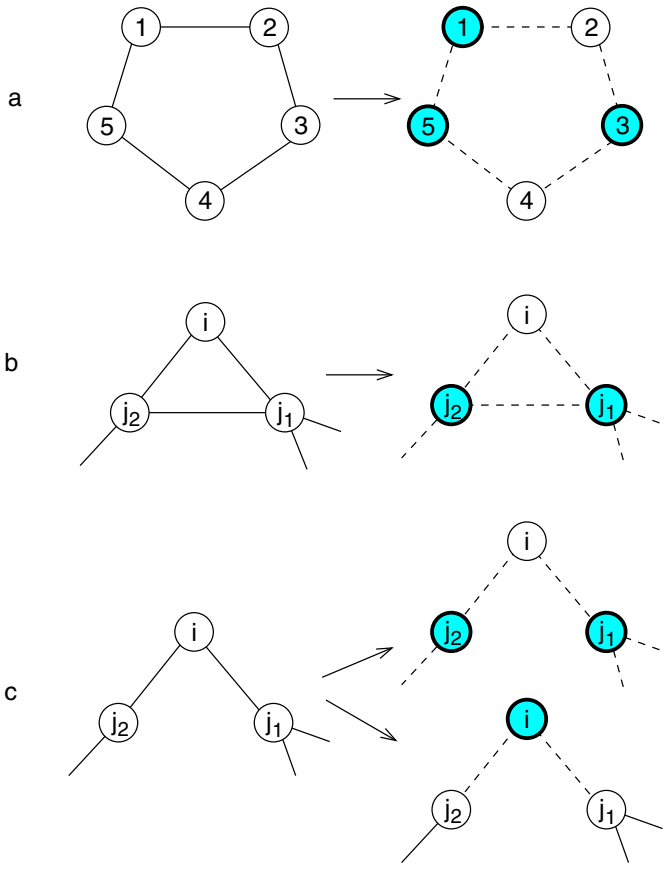


Figure 6.16: Algorithm of Tarjan and Trojanowski, case where the minimum degree $d_{\min} = 2$. Two cases are possible: all degrees are 2 (a) or there is at least one vertex with higher degree (b+c).

6.6 Monte Carlo simulations

In this section, we will introduce another method, called *Monte Carlo* (MC) *simulation*, which is a general simulation approach used to study computationally models in statistical physics. Here we are able to give only a short introduction, good and complete text books on this subject were written, e. g., by Newmann and Barkema [16], or by Landau and Binder [17]. When applied to VC, MC simulations allow for the calculation of approximations of minimum vertex covers. For Erdős–Rényi random graphs, the method does usually very well and one is able to find true minimum vertex covers for *all* connectivities, at least for sizes where a comparison with exact results is possible.

Although being able to calculate approximations of vertex covers, the nature of MC simulations is substantially different from the heuristics presented in Sec. 6.2. The basic approach is to interpret the graph and its covers as a configurations of a physical system, a *hard-core lattice gas*, the exact definition is given below. Then the system is evolved by MC simulations under the rules of statistical mechanics in the *grand-canonical ensemble* characterized by a *chemical potential* μ. By increasing μ continuously, or performing simulations at different values of μ in parallel (i. e., using *parallel tempering*, see below), one can obtain minimum-size vertex covers. The fundamental difference from the heuristics presented before is that the simulation does not run in one linear sweep, but many iterations have to be performed. To obtain true minimum vertex covers, one has to tune some parameters, e. g., the number of iterations and the number of different values for the chemical potential, which determine the overall computer time. Now we will present the details steps by step. First we introduce the hard-core gas, then we explain *Markov chains*, which provide the theoretical background for Monte Carlo simulations, we then discuss the algorithm for the case of the hard-core gas, and at the end we explain how parallel tempering works.

6.6.1 The hard-core lattice gas

For the definition of the hard-core gas on a given graph $G = (V, E)$, we consider arbitrary covers V_{vc}, including those of larger magnitude than the minimum vertex cover. This means all edges have to be *covered*, i. e., at least at one end-point of any edge there is a covering mark. Now we define the *uncovered* vertices as *occupied* by *particles* of the gas on G. Since edges having both neighbors being *uncovered* are prohibited, it is not allowed by definition that both end-points of any edge are occupied by particles. This can be interpreted as the particles having a *chemical radius* of one, i. e., they exhibit a hard-core repulsion that prevents them from coming too close to each other. An example of a hard-core gas is depicted in Fig. 6.17. Note that the trivial cover, i. e., choosing $V_{\mathrm{vc}} = V$, corresponds to having an empty configuration without any particle. For a detailed discussion of this model see the analytical approach in the next chapter. There the equivalence between vertex covers and particle packings will be exploited to use statistical-mechanics methods in an analytic description of VCs over random graphs.

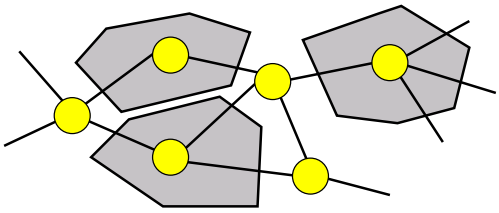


Figure 6.17: In an arbitrary vertex cover, every uncovered vertex can be seen as the position of a hard particle of chemical radius one. Due to the vertex cover constraint, no particles are allowed to overlap, i. e., they cannot occupy neighboring vertices.

6.6.2 Markov chains

The aim is to generate configurations of a given system, such that the generated configurations are distributed according to some given distribution. Here we want the hard-core gas to be sampled according to the grand-canonical distribution with chemical potential μ. Now we will describe the general formalisms of Markov Chains which can be used to perform this task.

The starting point is a system with a finite number of configurations $\{\mathbf{y}\}$ and given probabilities $P(\mathbf{y})$. Very often, as in the case of the hard-core gas, the configurations are vectors describing the states of a number N of vertices. Hence, the number of possible configurations is usually exponentially large in N. This means that typically $P(\mathbf{y}) \in \mathcal{O}(1)$ only for a few configurations, but the probability is exponentially small in N for most configurations. This will be an obstacle for a simple algorithm, as we will see.

The aim is the measurements of averages of observable quantities $A(\mathbf{y})$

$$\langle A \rangle := \sum_{\mathbf{y}} A(\mathbf{y}) P(\mathbf{y}) \,, \tag{6.8}$$

e. g., the average density of the hard-core gas. Since the number of configurations is assumed to be exponentially large in the number of degrees of freedom, a pure enumeration of all configurations is not feasible.

The most basic approach is to generate a certain number L of configurations $\{\mathbf{y}^i\}$ randomly, such that all configurations are equiprobable. Then we have:

$$\langle A \rangle \approx \overline{A}^{(a)} \equiv \sum_{\mathbf{y}^i} A(\mathbf{y}^i) P(\mathbf{y}^i) / \sum_{\mathbf{y}^i} P(\mathbf{y}^i) \,.$$

The equiprobable generation of the configurations is normally very easy. The problem is that usually the probability $P(\mathbf{y})$ is exponentially small, for almost all configurations, as already mentioned, hence the result $\overline{A}^{(a)}$ is very inaccurate. For the case of the hard-core gas, one can generate configurations independently and equiprobably by selecting the state *occupied/unoccupied* of each vertex independently, and then having $P(\mathbf{y}) = 0$ if the hard-core constraint is violated. This is obviously inefficient, since many configurations do not contribute at all.

It is better to generate the configurations in such a way that more important configurations, those with large probability, occur more often. This is called *importance sampling*. In the ideal case, this means that a number L of configurations $\{\mathbf{y}^i\}$ are generated independently such that they are immediately distributed according $P(\mathbf{y}^i)$. Then we can approximate the average directly by an arithmetic mean:

$$\langle A \rangle \approx \overline{A}^{(b)} \equiv \sum_{\mathbf{y}^i} A(\mathbf{y}^i) / L \,. \tag{6.9}$$

This direct way of generating random objects works in only few simple cases, e. g., when generating random numbers according to a Gaussian distribution. Unfortunately, such algorithms do not exists for most interesting models exhibiting many interacting degrees of freedom.

One way out of this problem is to use a *probabilistic dynamic* which generates a sequence (or chain) $\mathbf{y}(t)$ of configurations at discrete times $t = 0, 1, 2, \ldots$: $\mathbf{y}(0) \to \mathbf{y}(1) \to \mathbf{y}(2) \to \ldots$. We assume that configuration $\mathbf{y}(t+1)$ depends only on a (pseudo) random number and on the previous configuration $\mathbf{y}(t)$ in the chain. In this case $\{\mathbf{y}(t)|t = 0, 1, 2, \ldots\}$ is called a *Markov chain*. We describe the transitions $\mathbf{y}(t) \to \mathbf{y}(t+1)$ by *transition probabilities* $W_{\mathbf{yz}} = W(\mathbf{y} \to \mathbf{z})$, i. e., the probability that the system moves from configuration $\mathbf{y}$ (at time t) to configuration $\mathbf{z}$ (at time $t+1$). For simplicity we assume furthermore that $W_{\mathbf{yz}}$ does not depend on the time t explicitly. The transition probabilities have the following properties:

$$\begin{aligned} W_{\mathbf{yz}} &\geq 0 \quad \forall \mathbf{y}, \mathbf{z} \quad (\textit{positivity}) \\ \sum_{\mathbf{z}} W_{\mathbf{yz}} &= 1 \quad \forall y \quad (\textit{conservation}) \,. \end{aligned} \tag{6.10}$$

The configuration space together with the transition probabilities is called a *Markov process*. Now we analyze the Markov process by introducing $P(\mathbf{y}, t)$, which is the the probability that the Markov process is at time t in configuration $\mathbf{y}(t) = \mathbf{y}$. We describe the change of the probability to be in state $\mathbf{y}$ when going from time t to $t+1$. For this purpose, we have to consider all transitions which go out of configuration $\mathbf{y}$, i. e., which decrease $P(\mathbf{y}, t+1)$, and the transitions which go into $\mathbf{y}$ from other configurations, i. e., which increase $P(\mathbf{y}, t+1)$, see Fig. 6.18.

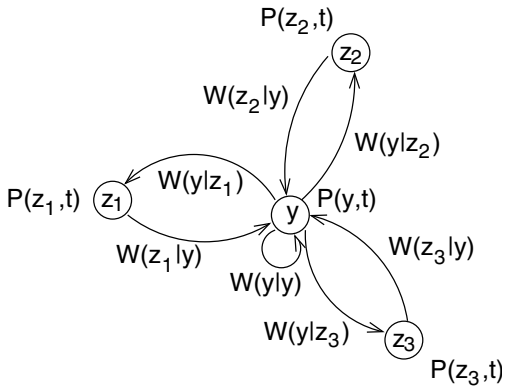


Figure 6.18: The change of the probability of being in configuration y in a Markov process is determined by the transitions into and out of the "neighboring" configurations and the probabilities of being in the "neighboring" configurations. An example with four configurations. Transitions having transition probability zero are not shown.

The change of probability for configuration $\mathbf{y}$ is then given by the *master equation*

$$\Delta P(\mathbf{y}, t) := P(\mathbf{y}, t+1) - P(\mathbf{y}, t) = \sum_{z} W_{\mathbf{zy}} P(\mathbf{z}, t) - \sum_{z} W_{\mathbf{yz}} P(\mathbf{y}, t) \quad \forall \mathbf{y}. \tag{6.11}$$

Under certain circumstances (e. g., if there is only one eigenvalue $\lambda = 1$ for the matrix $(W_{\mathbf{yz}})$, see [18]), the probability distribution $P(\mathbf{y}, t)$ converges towards the *stationary* (time-independent) distribution

$$P_{ST}(\mathbf{y}) \equiv \lim_{t\to\infty} P(\mathbf{y}, t)\,.$$

It is independent of the starting configuration $\mathbf{y}(0)$. Such a system is called *ergodic*. Ergodicity means that there exists between any two configurations a path with all transitions along the path having non-zero transition probabilities. Hence, one can reach each configuration from all other configurations.

Now we want to set up the transition probabilities, such that the stationary distribution is the distribution P we want to have:

Target: Choose $W_{\mathbf{yz}}$ such that $P_{ST} = P$

Since $P(.)$ is time-independent, it follows from Eq. (6.11):

$$0 = \Delta P(\mathbf{y}) = \sum_z W_{\mathbf{zy}} P(\mathbf{z}) - \sum_{\mathbf{z}} W_{\mathbf{yz}} P(\mathbf{y}) \quad \forall \mathbf{y}\,.$$

This is a third type of condition for the transition probabilities. This condition means that the total change in time of the probability for each configuration is zero, i. e., the "flows" of probability in and out of the configurations balance out. There are many ways to fulfil this condition. One way is to request that the balance holds for all pairs of configurations, i. e.,

$$W_{\mathbf{zy}} P(\mathbf{z}) - W_{\mathbf{yz}} P(\mathbf{y}) = 0 \quad \forall \mathbf{y}, \mathbf{z}\,. \tag{6.12}$$

This condition is called *detailed balance*.

By obeying detailed balance, and having an ergodic system, it is guaranteed that the Markov process generates configurations, which are distributed according to $P(.)$ in the long-time limit. Hence one can take averages as in Eq. (6.9). Note that the stationary distribution is obtained formally only in the case $t \to \infty$. This means for practical situations that the configurations at the beginning of the Markov chain depend strongly on the initial configuration $\mathbf{y}(0)$. Therefore, one usually omits the first $t < t_{\mathrm{eq}}$ configurations from the calculation of averages. One says, the system has to be *equilibrated*. A suitable choice for t_{eq} usually has to be determined within the simulation of the model under investigation, e. g. one can measure correlations with $\mathbf{y}(0)$, or start two simulations with two atypical, strongly different initial configurations and wait till the measurable quantity one is interested in has converged in both cases to the same value, for details see [16, 17]. Furthermore $\mathbf{y}(t+1)$ is usually strongly correlated with $\mathbf{y}(t)$. Hence, only "distant" configurations $\mathbf{y}(t), \mathbf{y}(t+\Delta t), \mathbf{y}(t+2\Delta t), \ldots$ exhibit a small statistical correlation. Again Δt has to be determined empirically for each model, for each observable and for the current parameters.

6.6.3 Monte Carlo for hard-core gases

Now we present transition rules for the hard-core gas, such that the stationary distribution is the grand-canonical distribution. We describe a configuration by the vector $\underline{\nu} = \{\nu_i\}$ $(i \in V)$, with

$$\nu_i = \begin{cases} 1 & \text{for } i \text{ is occupied by a particle} \\ 0 & \text{else .} \end{cases} \tag{6.13}$$

To distinguish valid configurations from those where the hard-core constraint is violated, we use the following indicator function:

$$\chi(\underline{\nu}) = \prod_{\{i,j\}\in E} (1 - \nu_i \nu_j) \,. \tag{6.14}$$

The function χ takes value one whenever $\underline{\nu}$ corresponds to an allowed particle packing (or a VC) because all factors equal one. Whenever there are two neighboring vertices occupied by particles, i. e., $\nu_i = \nu_j = 1$, the value of χ equals zero.

Thus, the grand-canonical distribution is given by

$$P(\underline{\nu}) = \frac{1}{\Xi} \chi(\underline{\nu}) e^{\mu \sum_i \nu_i} \tag{6.15}$$

where Ξ is the grand-canonical partition function, i. e., the normalization constant (for details cf. Chap. 7), μ the chemical potential and $\sum_i \nu_i$ evaluates the number of particles. In this distribution, packings of different numbers of particles are allowed, weighted according to these numbers. The system is said to be coupled to a particle bath. This non-constant particle number is important for our purpose, because we do not know the size X_c of a minimum VC before having constructed one.

For $\mu > 0$, configurations of higher particle number have higher statistical weight. This means that for $\mu \to \infty$, configurations with the highest density $\rho = \frac{1}{N} \sum_i \nu_i$ are obtained. The number of *unoccupied* vertices is minimized, hence a minimum vertex cover is obtained.

The MC simulations consist of two types of transition: the "move" (M) transition and the "exchange" (E) transition. For each MC step, either the M or the E transition is performed, each with probability $1/2$. For both types of step, a vertex i is selected randomly with probability $1/|V|$. The transitions proceed as follows, see also Fig. 6.19.

M If the vertex is not *occupied* by a particle ($\nu_i = 0$), *and* if exactly one neighboring vertex j is *occupied* ($\nu_j = 1$), then the particle at j is moved to vertex i ($\nu_i \leftarrow 1$, $\nu_j \leftarrow 0$). In all other cases, nothing happens, i. e., the configuration remains unchanged.

E If vertex i is *unoccupied* and *all* neighbors are also *unoccupied*, then a particle is inserted on vertex i. If some neighbors are occupied, the configuration remains unchanged.

If vertex i is *occupied*, the particle is removed with probability $\exp(-\mu)$. This means a (pseudo) random number r is drawn which is uniformly distributed in the interval $[0, 1]$. If $r < \exp(-\mu)$, then the particle is removed, otherwise not.

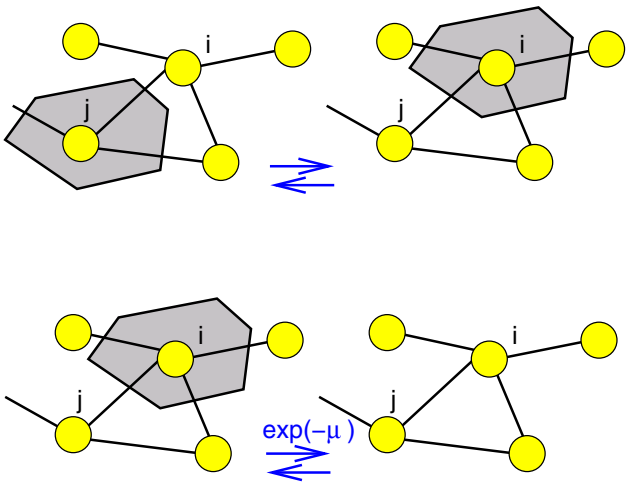


Figure 6.19: Two type of move are used for the hard-core lattice gas simulation. Top: particles are translated to neighboring vertices (move). Bottom: particles are inserted into/removed from the lattice (exchange).

Now, we want to prove that the algorithm fulfils detailed balance. Since the M and E transitions are independent of each other, it is sufficient to show that both types of transition fulfil detailed balance separately. The cases where the configuration does not change, obey detailed balance by definition, because for $\underline{\xi} = \underline{\zeta}$, Eq. (6.12) is fulfilled trivially. Therefore, we only have to consider the cases, where $\underline{\xi} \neq \underline{\zeta}$ and the M or the E transition is possible between the two configurations.

M Let $\underline{\xi}, \underline{\zeta}$ be two valid configurations (i. e., $\chi(\underline{\nu}) = \chi(\underline{\zeta}) = 1$), which are the same except $\xi_i = 0, \xi_j = 1$ and $\zeta_i = 1, \zeta_j = 0$. For the M transition, the move $\underline{\xi} \to \underline{\zeta}$ is performed if vertex i is selected, i. e., with probability $W_{\underline{\xi}\underline{\zeta}} = 1/|V|$. Similarly, the move $\underline{\zeta} \to \underline{\xi}$ is performed if vertex j is selected, i. e., $W_{\underline{\zeta}\underline{\xi}} = 1/|V| = W_{\underline{\zeta}\underline{\xi}}$. Since the number of particles does not change by this transition, we have trivially $P(\underline{\xi}) = P(\underline{\zeta})$. It follows $W_{\underline{\zeta}\underline{\xi}} P(\underline{\zeta}) - W_{\underline{\xi}\underline{\zeta}} P(\underline{\xi}) = 0$, i. e., detailed balance is fulfilled according to Eq. (6.12).

E Let $\underline{\xi}, \underline{\zeta}$ be two valid configurations, which are the same except $\xi_i = 0$ and $\zeta_i = 1$. For the E transition, $W_{\underline{\xi}\underline{\zeta}} = 1/|V|$ and $W_{\underline{\zeta}\underline{\xi}} = \exp(-\mu)/|V|$. Furthermore we have

$$\begin{aligned} P(\underline{\xi}) &= Z^{-1}\chi(\underline{\xi}) \exp\left(\mu \sum_i \xi_i\right) \\ &= Z^{-1}\chi(\underline{\xi}) \exp(-\mu) \exp\left(\mu\left(1 + \sum_i \xi_i\right)\right) \\ &= Z^{-1}\chi(\underline{\zeta}) \exp(-\mu) \exp\left(\mu \sum_i \zeta_i\right) \\ &= \exp(-\mu) P(\underline{\zeta})\,. \end{aligned}$$

Hence, we get $W_{\underline{\zeta}\underline{\xi}}P(\underline{\zeta}) - W_{\underline{\xi}\underline{\zeta}}P(\underline{\xi}) = \frac{1}{|V|}\exp(-\mu)P(\underline{\zeta}) - \frac{1}{|V|}\exp(-\mu)P(\underline{\zeta}) = 0$, i. e., detailed balance is fulfilled according to Eq. (6.12).

The transition rates fulfilling detailed balance are not uniquely determined. It is possible to invent other types of transition than the M and E transitions explained above. Note that the M transition is defined on the basis of the *unoccupied* vertices. When defining another "move" transition on the basis of the *occupied* vertices, the possible number of *unoccupied* neighbors, to where the particle can move, might be different for two neighboring vertices i, j. Hence, to ensure detailed balance, one would have to include these numbers in the transition probabilities.

The simulation is performed in terms of *Monte Carlo sweeps*. One sweep for a graph of $N = |V|$ vertices consist of exactly N tried transitions, each time randomly selecting a vertex and then, with equal probabilities, trying either an M or an E transition.

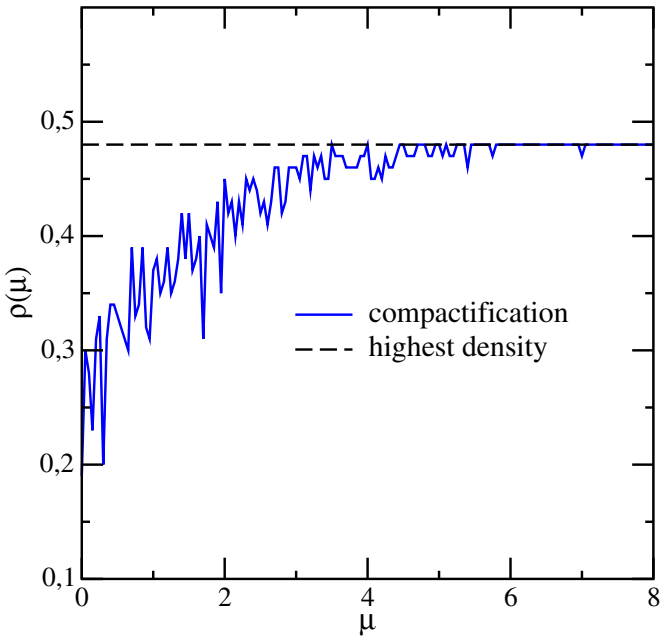


Figure 6.20: Compactification using MC simulations of a hard-core gas on a sample random graph of size $N = 100$ for average degree $c = 4.0$. Density ρ as a function of chemical potential μ for one single run. The highest possible density, corresponding to minimum vertex covers of this graph, are indicated by a horizontal line.

As an example of the application of the MC method, we show in Fig. 6.20 the *compactification* of a hard-core gas on a random graph of size $N = 100$ with connectivity $c = 4.0$. The simulation is started at $\mu_i = 0$, then μ is gradually increased by $\delta\mu = 0.05$ up to $\mu_f = 8$. For each value of μ, 10 MC sweeps are performed, and the density ρ after the 10th sweep is recorded. As expected, the density has a tendency to increase with the value of μ. The function

is not smooth, because only the data for one single run is shown. When averaging over many independent runs, $\rho(\mu)$ would be a smooth monotonically increasing function. Interestingly, the MC simulation is able to find the configurations with the highest densities, corresponding to minimum vertex covers.

Note that the quality of the MC compactification depends on the ensemble of graphs. If one considers, e. g., random graphs made of randomly joined tetrahedrons, i. e., of cliques of size 4, then the corresponding system of hard-core particles behaves in a *glassy* way. This means the algorithm gets stuck in meta-stable configurations which are very different from the true ground states [19]. The reason for this behavior is that the configuration space has a very complicated and rugged structure. The meta-stable configurations have large but not maximum densities, and they are surrounded by configurations of lower densities. This means that, at high values of the chemical potential, the simulation will stay in the meta-stable configurations. In this case, an enhanced version of the MC method, the *parallel tempering*, may help. This approach is outlined in the following section.

6.6.4 Parallel tempering

The basic idea of parallel tempering [20, 21], is to simulate several copies of the same system, but with possibly different configurations, kept at different values $\mu_1 < \mu_2 < \ldots < \mu_n$ of the chemical potential, see Fig. 6.21. The M and E moves introduced above act locally on all configurations at the different values of the chemical potential. For parallel tempering, a *swap* (S) transition is additionally introduced, which tries to exchange the configurations having any two neighboring values of the chemical potential μ_k, μ_{k+1} ($k \in [1, n-1]$). This allows the configurations to be subjected to different values of μ during the simulation. Hence a configuration being stuck in a meta-stable state at a high value of μ, might be simulated later on with considerably smaller values of the chemical potential, i. e., at much lower expected density. When this configuration again visits high values of μ, it might explore a different region in configuration space, where it possibly can reach higher densities than before.

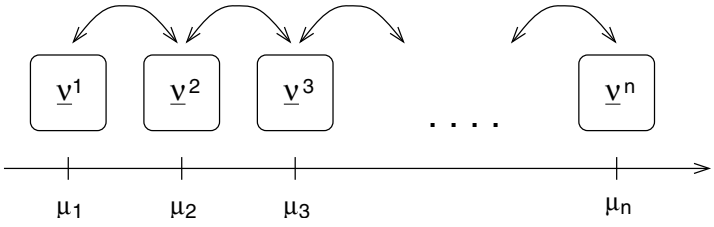


Figure 6.21: Parallel tempering with n different values of the chemical potential $\mu_1 < \mu_2 < \ldots < \mu_n$. At each value μ_i a system is simulated using conventional MC. From time to time, configurations are exchanged between neighboring values μ_k, μ_{k+1}, such that detailed balance is fulfilled for the combined systems.

The key point is that the S transition must be implemented in a way that detailed balance still holds, although two configurations being held at two different values of the chemical potential are treated. For an S transition, first one chooses a value $k \in \{1, 2, \ldots, n-1\}$ randomly. Let $\underline{\xi}, \underline{\zeta}$ be the configurations being simulated currently at values μ_k,μ_{k+1}. The joint probability for both configurations in the grand-canonical ensemble, which is the crucial quantity to be taken into account here, is given by

$$P_{k,k+1}(\underline{\xi}, \underline{\zeta}) = \frac{1}{\tilde{\Xi}_{k,k+1}} \chi(\underline{\xi}) \exp\left(\mu_k \sum_i \xi_i\right) \chi(\underline{\zeta}) \exp\left(\mu_{k+1} \sum_i \zeta_i\right), \tag{6.16}$$

with $\tilde{\Xi}_{k,k+1}$ being the corresponding partition function and the dependency on the chemical potential indicated by the index $k, k+1$. To define the transition probabilities, we evaluate the quantity

$$\Delta_{k,k+1}(\underline{\xi}, \underline{\zeta}) \equiv (\mu_k - \mu_{k+1}) \left(\sum_i \xi_i - \sum_i \zeta_i\right) \tag{6.17}$$

and choose

$$W_{k,k+1}([\underline{\xi}, \underline{\zeta}] \rightarrow [\underline{\zeta}, \underline{\xi}]) = \exp(-\max[\Delta_{k,k+1}(\underline{\xi}, \underline{\zeta}), 0]) \,. \tag{6.18}$$

The swap does not take place with the probability $1 - W_{k,k+1}([\underline{\xi}, \underline{\zeta}] \rightarrow [\underline{\zeta}, \underline{\xi}])$. This definition of the transition probability means that when at the larger value μ_{k+1} the configuration $\underline{\zeta}$ has a lower density than $\underline{\xi}$ (which is at the smaller value μ_k), i. e., when one encounters an atypical situation, we get $\Delta(\underline{\xi}|\mu_k, \underline{\zeta}|\mu_{k+1}) < 0$. This results in a transition rate 1, i. e., the swap will be performed always in this case, hence transitions to typical situations a favored. Note that $\Delta_{k,k+1}(\underline{\xi}, \underline{\zeta}) = -\Delta_{k,k+1}(\underline{\zeta}, \underline{\xi})$.

To prove detailed balance, we assume w.l.o.g. $\sum_i \xi_i - \sum_i \zeta_i \geq 0$, hence we obtain $\Delta_0 \equiv \Delta_{k,k+1}(\underline{\xi}, \underline{\zeta}) < 0$ leading to $W_{k,k+1}([\underline{\xi}, \underline{\zeta}] \rightarrow [\underline{\zeta}, \underline{\xi}]) = 1$ for one direction of the transition and $W_{k,k+1}([\underline{\zeta}, \underline{\xi}] \rightarrow [\underline{\xi}, \underline{\zeta}]) = \exp(\Delta_0)$ for the inverse transition. This leads to (omitting $\chi(\underline{\xi}) = \chi(\underline{\zeta}) = 1$)

$$\begin{aligned}
& W_{k,k+1}([\underline{\xi}, \underline{\zeta}] \rightarrow [\underline{\zeta}, \underline{\xi}]) P_{k,k+1}(\underline{\xi}, \underline{\zeta}) - W_{k,k+1}([\underline{\zeta}, \underline{\xi}] \rightarrow [\underline{\xi}, \underline{\zeta}]) P_{k,k+1}(\underline{\zeta}, \mathbf{k}) \\
= \;& 1 P_{k,k+1}(\underline{\xi}, \underline{\zeta}) - \exp(\Delta_0) \, P_{k,k+1}(\underline{\zeta}, \underline{\xi}) \\
= \;& 1 \frac{1}{\tilde{Z}_{k,k+1}} \exp\left(\mu_k \sum_i \xi_i\right) \exp\left(\mu_{k+1} \sum_i \zeta_i\right) - \\
& \exp\left((\mu_k - \mu_{k+1})(\sum_i \xi_i - \sum_i \zeta_i)\right) \frac{1}{\tilde{Z}_{k,k+1}} \exp\left(\mu_k \sum_i \zeta_i\right) \exp\left(\mu_{k+1} \sum_i \xi_i\right) \\
= \;& 0 \,.
\end{aligned} \tag{6.19}$$

Therefore, detailed balance is fulfilled according to Eq. (6.12).

Within a parallel tempering simulation one has to deal with several parameters: one has to determine the range of chemical potentials, its number n, and the precise values μ_k. Usually it is efficient to have more values in the region of high chemical potentials, and only few

values for small chemical potentials. One possibility is to choose the values of the chemical potential automatically by an iterative procedure, such that the transition probability is 0.5 for all swaps [21]. Also one has to choose how often a "swap" step is performed in comparison with the local M and E transitions. One can, e. g., perform one MC sweep for each copy and then try $n-1$ "swaps". Another parameter is the total number of iterations.

Indeed, using parallel tempering, for larger random graph of size, e. g., $N = 1000$, higher density states can be found compared with simple compactification [22]. Also higher density configurations for "glassy" ensembles can be obtained.

6.7 Backbone

Minimum vertex covers may not be unique. In the case where several vertex covers $V_{\rm vc}^{(1)}, \ldots, V_{\rm vc}^{(K)}$ exist, each with the same cardinality X (not necessarily minimum vertex covers), a vertex i is called a *backbone* vertex, if it is either a member of all covers ($\forall k = 1, \ldots, K : i \in V_{\rm vc}^{(k)}$) or else a member of no cover ($\forall k = 1, \ldots, K : i \notin V_{\rm vc}^{(k)}$). These vertices can be regarded as *frozen* in all vertex-cover configurations. All other vertices are called *non-backbone* vertices.

Example: Minimum vertex cover

For the graph from Fig. 6.1, in Fig. 6.22 all three minimum vertex covers ($X_{\rm c} = 3$) are shown. Vertices 2 (always *uncovered*) and 3 (always *covered*) belong to the backbone, while all other vertices are non-backbone.

It is straightforward to show that vertex 3 must be a member of all minimum vertex covers. Assume that vertex 3 is not *covered*. Since all edges have to be *covered*, all neighbors of vertex 3 have to be *covered*. Thus, vertices $1, 2, 4$ and 5 have to be *covered*, i. e., more vertices than in the minimum vertex cover, which has size $X_{\rm c} = 3$.

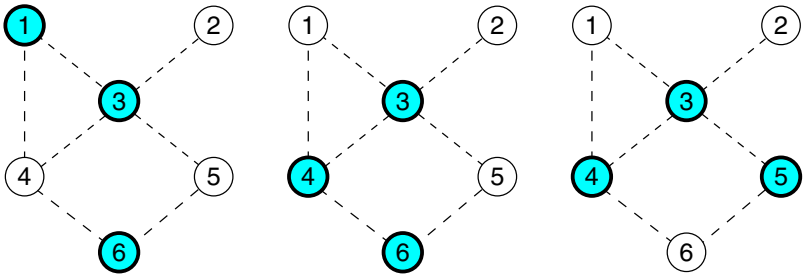


Figure 6.22: All three minimum vertex covers of the graph from the preceding example.

□

One way to obtain the backbone of a given graph is to enumerate all solutions. All vertices having the same state in all configurations belong to the backbone B. In this case the branch-and-bound algorithm must be changed. In the version above, only subtrees of the configuration tree were visited, where better solutions are possible. Now the algorithm has to continue also in subtrees where equally good solutions may be found. The number of steps needed by this algorithm is at least as large as the number of solutions, i. e., physically speaking, as large as the degeneracy of the system.

If one is interested only in the backbone, one should note that for $x > x_{\mathrm{c}}(c)$ the backbone is zero in the thermodynamic limit. The argument follows simply by contradiction. Assume first that a vertex i is in the, always uncovered, backbone. Than take a minimum VC of size $x_{\mathrm{c}}N$. The graph is covered, and covering marks are left. They can be distributed among the uncovered vertices, in particular we can guarantee i to be covered, which is a contradiction to the assumption. If we assume, on the other hand, that the vertex is in the covered backbone, we proceed in a similar fashion. We take one minimum VC. Then, almost surely, the degree of i is finite, i. e., smaller than the number $(x - x_{\mathrm{c}})N$ of available covering marks. We cover all neighbors, uncover i, and distribute the other covering marks, producing thus a VC of cardinality xN with i being uncovered. This is in contradiction to the second assumption, the backbone is thus almost surely empty for $x > x_{\mathrm{c}}(c)$.

If one is interested in the backbone B of the minimum covers, one can avoid the enumeration of *all* minimum covers. The basic idea is to start with the calculation of *one* minimum cover V_{vc}^1. Now the fact is used that the backbone consists of vertices which are always covered (ac) and of vertices which are always uncovered (auc), i. e., $B = B^{ac} \cup B^{auc}$. First the ac backbone B^{ac} is determined. Hence, we have to search only among vertices, which are *covered* in the first cover, i. e., the members of V_{vc}^1. We consider a vertex $i \in V_{\mathrm{vc}}^1$. It is in the ac backbone, if excluding it from being *covered* increases the size of a cover. Hence, if $d_i = 1$, vertex i has only one neighbor, then we can cover the neighbor instead without increasing the cover size, i. e., $i \notin B^{ac}$. If $d_i \geq 2$, we create a modified graph G' by replacing i by d_i new vertices, each one connects to *one* edge which was incident to i in the original graph. Now we calculate a minimum cover again. First, please note that if one just covers all $d_i > 1$ new vertices instead of vertex i in the original graph, then the size of the cover is increased. This is probably not the minimum cover of G'. If i is not in the backbone for the original graph, it is possible to rearrange the cover in the original graph, such that i is not covered. This means that, in the modified graph, one can find a cover having all new nodes *uncovered* and the minimum cover must be of the same size $|V_{\mathrm{vc}}^1|$. On the other hand, if i is in the ac backbone, then it is very "important" in the original graph, such that more than one vertex must take its role in the modified graph, hence the size of the minimum cover must increase.

By iterating over all vertices from V_{vc}^1, the ac backbone can be found. Now the determination of the auc backbone is simple. For each auc backbone vertex i, all edges incident to it must be covered in all vertex covers, while i is never covered. This means that all neighbors of i must be always covered, hence all neighbors must belong to the ac backbone.

The algorithm can be summarized as follows:

algorithm backbone(G)
begin
  calculate one min cover $V_{\rm vc}^1$ of G

  comment calculate always-covered backbone:
  for all $i \in V_{\rm vc}^1$ **do**
  if degree $d_i = 1$ **then**
    i is not in the backbone
  else
  begin
    create new graph G':
      remove i
      for each dangling end of edges **do**
        add one vertex at the dangling end
    calculate min cover $V'_{\rm vc}$ of G'
    if $|V'_{\rm vc}| > |V_{\rm vc}^1|$ **then**
      i is in backbone
    else
      i is not in backbone
  end

  comment Calculate always-uncovered backbone vertices:
  for all vertices i which have only ac backbone neighbors:
    i is in backbone (uncovered)
  comment also vertices with degree 0 are backbone (uncovered)
end

Example: Backbone algorithm

This example demonstrates how the backbone-finding algorithm operates. We again consider the graph treated in Fig. 6.22.

First *one* minimum cover is calculated. The result can be, for example, the minimum cover $V_{\rm vc}^1$ presented in the left part of Fig. 6.22, it is again shown in Fig. 6.23. Vertices 1, 3 and 6 are candidates for the ac backbone. The resulting modified graph G' for vertex 1 is shown in the middle of Fig. 6.23: Vertex 1 has degree 2, hence it is replaced by two vertices 1a,1b. The size of the minimum cover is equal to the size of the minimum cover of the original graph, hence vertex 1 does not belong to the ac backbone. Note that the minimum cover of G' corresponds to the minimum cover which is shown in the middle of Fig. 6.22. In the same way can be shown that vertex 6 does not belong to the ac backbone.

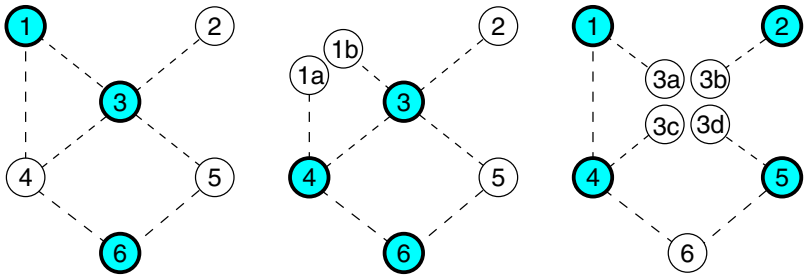


Figure 6.23: Example of how the backbone-finding algorithm operates. In the left part the graph with one cover $V_{\rm vc}^1$ is shown. In the middle part, it is tested whether vertex 1 is part of the ac backbone. Since the minimum-cover size of the modified graph is equal to $|V_{\rm vc}^1| = 3$, vertex 1 is not an ac backbone vertex. In the right part it is tested whether vertex 3 is part of the ac backbone. Here the size of the minimum-cover of the modified graph is larger than in the original graph, hence $3 \in B^{ac}$.

In the right part of Fig. 6.23, the modified graph for vertex 3 is shown along with a minimum cover. Its size is larger than $|V_{\rm vc}^1|$, hence vertex 3 belongs to the ac backbone B^{ac}.

Hence vertex 3 is the only member of B^{ac}. Since vertex 2 is the only vertex which has only members in B^{ac}, we obtain $B^{auc} = \{2\}$. Please compare the result of the backbone-finding algorithm with the result from enumerating all minimum covers shown in Fig. 6.22. □

Note that for the branch-and-bound algorithm, the typically hardest instances are found directly at the phase transition. Hence, calculating the backbone always take exponential time, even if it is not necessary to enumerate all (exponentially many) solutions. If, however, the leaf-removal algorithm presented before is working, i. e., if it outputs a minimum VC, then also determining becomes polynomial. This is true, in particular, for random graphs of average degree $c < e$.

Now, we are interested in the resulting fraction $b_{\rm c}(G) = |B|/N$ of backbone vertices. This quantity again is averaged by considering different realizations of the random graphs, for one graph size N. The process is performed for different values of N. These data can be used to extrapolate to the thermodynamic limit via a finite-size scaling function. The result, as a function of different average degrees c, is displayed in Fig. 6.24 and compared with the analytical result to be derived in the following chapter of this book. Again, a very good agreement can be observed for low values $c < e \approx 2.718$, while for graphs having a higher degree, larger deviations occur. Note that for the case $c = 0.0$, where the graph has no edges, no vertex needs to be covered, meaning that all vertices belong to the backbone [$b_{\rm c}(0) = 1$].

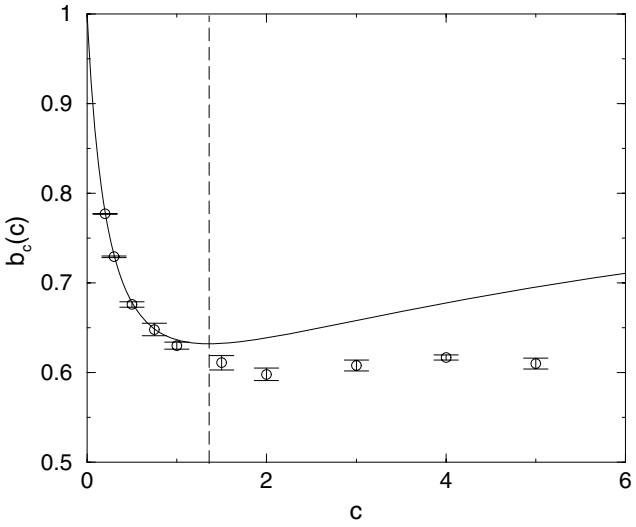


Figure 6.24: The total backbone size of minimum vertex covers as a function of c. The solid line shows the analytical result. Numerical data are represented by the error bars. They were obtained from finite-size scaling fits similar to the calculation for $x_{\mathrm{c}}(c)$. The vertical line is at $c = e \approx 2.718$ where the analytical results cease to be exact.

6.8 Clustering of minimum vertex covers

Apart from identifying backbone and non-backbone vertices, one can analyze the organization of minimum vertex covers in more detail. One possible approach is to look for clusters of solutions [22].

Usually the minimum vertex covers are not equally distributed over the configuration space. They cluster in one or many groups that are separated by regions where no cover exists, or where the covers have larger size. Understanding this organization, it is possible to discover many interesting things about the nature of the model, and possibly also about its computational hardness.

Such clustering effects have already been observed in statistical physics for spin glasses [26, 27]. For the mean-field Ising spin glass, also called the Sherrington–Kirkpatrick (SK) model [28], which is defined on a complete, i. e., fully connected graph, Parisi has constructed [29] an analytical solution for the free energy. This solution exhibits so-called replica symmetry breaking (RSB), which means that the state space is organized in an infinitely nested hierarchy of clusters of states, characterized by ultrametricity [31]. Recently, this solution for the free energy was mathematically proven to be the exact one [32]. Also in numerical studies the clustering structure of the SK model has been observed, e. g., by calculating the distribution of overlaps [33–35], by studying the spectrum of spin–spin correlation matrices [36,37], or by directly applying clustering algorithms [38]. For finite-dimensional spin glasses, RSB seems

not to be fully present [39,40] (at least not in the same way as for the mean-field model), since clustering has been observed numerically, but it is not non-ultrametric [38]. On the other hand, for models like Ising ferromagnets it is clear that they do not exhibit RSB and all solutions are organized in, at most, a few clusters which are related by the system's symmetries.

The use of the analytical tools from statistical mechanics enables physicists to contribute to the analysis of problems that originate in theoretical computer science. We will see in Chap. 7, how these approaches can be applied to VC. Usually, one can only calculate the solution in the case of replica symmetry [41, 42], or in the case of one-step replica symmetry breaking (1-RSB) [43–45], and look for the stability of the solutions. For this reason, the relation between the (partial) analytical solution and the clustering structure is not well established. It is far from being clear for most models how the clustering structure appears. However, most statistical physicists believe that the failure of replica symmetry (RS) leads indeed to clustering [46, 47]. On the other hand, it is unlikely that the clustering of models on dilute graphs, like the random graphs treated here, is exactly the same as is found for the mean-field SK spin glass. So, from the physicists point of view, it is quite interesting to study the organization of the phase space using numerical methods to understand better the meaning of "complex organization of phase space" for not fully connected models, such as combinatorial optimization problems. Here we will just present the result of one numerical study of the clustering properties of the vertex-cover problem.

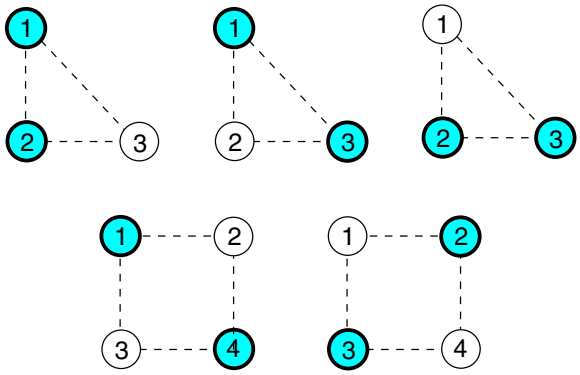


Figure 6.25: A triangle graph with 3 vertices (top) has 3 different minimum vertex covers, which are all neighbors and hence form one cluster. A square graph (bottom) exhibits two minimum clusters, which are not neighbors according to the definition used here.

Let us first define what we mean by the notation *cluster*. We call two minimum vertex cover configurations *neighbors*, if they differ by the states of exactly two vertices i, j. This means a covering mark has been moved from vertex i to vertex j. Since we are just interested in minimum covers, it follows immediately, that i and j are connected by an edge. A cluster is now the transitive closure of the neighbor relation. This means that two configurations belong to the same cluster, if one can move in configuration space from one to the other by just moving covering marks around, while always keeping the graph covered. This means that all

three solutions of a triangle graph belong to one cluster, while the two minimum vertex covers of a square graph, do *not* belong to the same cluster, see Fig. 6.25.

Note that one could define clusters in different ways, e. g., that one could consider configurations as neighbors if they can be reached by moving at most a finite fixed number $k_{\max}$ of covering marks. For $k_{\max} \geq 2$ the two minimum vertex covers of the square graph in Fig. 6.25 would also belong to the same cluster. Increasing $k_{\max}$ leads to the merging of previously separated clusters, $k_{\max} = 1$ gives the finest possible clustering. We study only the case $k_{\max} = 1$ here. Other ways of studying clustering exist, e. g., to study the spectrum of spin–spin correlation matrices [36, 37] or so-called direct clustering [38]. Since we are interested here in just giving one example of what can be learned from numerical clustering, we consider only the definition of a cluster given above.

For small systems, one can enumerate all minimum vertex covers, and perform the clustering by just testing pairwise whether configurations are neighbors. Note that for the numerics, it helps to identify first the backbone vertices, as explained in Sec. 6.7. Then one can remove all backbone vertices and the graph usually splits into smaller components, which then can be treated independently, which sometimes greatly reduces the running time.

For larger systems, such an enumeration is no longer possible, because the number of minimum vertex covers grows, in principle, exponentially with the system size. Furthermore, even obtaining exact minimum vertex covers becomes impossible if the system size is too large. In this case, one can obtain multiple vertex covers by performing several independent runs of

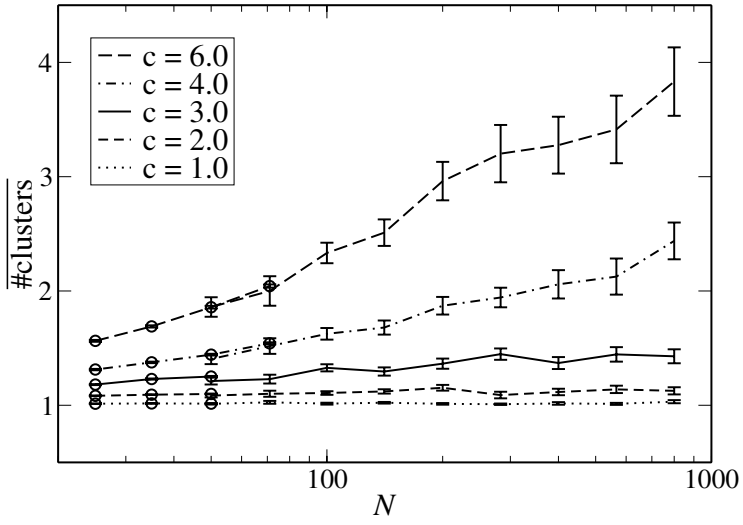


Figure 6.26: Average number of clusters in the solution space of the largest component as a function of system size. The circle symbols for small system sizes have been obtained by clustering complete sets of ground states. For large systems we sampled ground states with a parallel tempering algorithm at large but finite chemical potential μ.

parallel tempering MC simulation, as explained in Sec.6.6. Now, it does not make sense to look for direct neighbors, but one can apply a test, the *ballistic search* [48] which checks in a probabilistic manner whether two given configurations belong to the same cluster. Note that the minimum vertex covers of different connected components of a graph are independent of each other. We perform the clustering only for the solutions of the largest component.

Having performed the clustering, one interesting quantity to evaluate is the *number of clusters*. The reason is that for a simple structure of the configuration space, i. e., in the case of replica symmetry, one believes that only one cluster, or a small finite number of clusters exists in the thermodynamic limit. As we will see in Chap. 7, the solution of VC is replica symmetric for connectivities $c < e$. And indeed, as can be seen in Fig. 6.26, the cluster number stays finite and small for $c < e$. For larger connectivities, the number of clusters seems to grow logarithmically. This information is interesting, since it cannot be derived from analytical studies so far. More results on the clustering properties are presented in Ref. [22]. Here, we move on to the analytical approaches.

Bibliography

[1] K. Mehlhorn and St. Näher, *The LEDA Platform of Combinatorial and Geometric Computing* (Cambridge University Press, Cambridge 1999);
see also `http://www.mpi-sb.mpg.de/LEDA/leda.html`

[2] J. Siek, L.-Q. Lee, A. Lumsdainesee, *The Boost Graph Library*, (Addison-Wesley, Reading (MA) 2001);
see also `http://www.boost.org/libs/graph/doc/index.html`

[3] J. Hromkovic, *Algorithms for Hard Problems*, (Springer, Heidelberg, 2001).

[4] E. L. Lawler and D. E. Wood, Oper. Res. **14**, 699 (1966).

[5] R. Lüling and B. Monien, in: *Sixth International Parallel Processing Symposium*, (IEEE Comput. Soc. Press, Los Alamitos, USA 1992).

[6] M. Shindo and E. Tomita, Syst. Comp. Jpn. **21**, 1 (1990).

[7] A. K. Hartmann, W. Barthel and M. Weigt, submitted to Comp. Phys. Comm.

[8] M. N. Barber, in: C. Domb and J. L. Lebowitz, *Phase Transitions and Critical Phenomena* **8**, 146, (Academic Press, London 1983).

[9] V. Privman (ed.), *Finite Size Scaling and Numerical Simulation of Statistical Systems*, (World Scientific, Singapore, 1990).

[10] J. Cardy, *Scaling and Renormalization in Statistical Physics*, (Cambridge University Press, Cambridge 1996).

[11] P. G. Gazmuri, Networks **14**, 367 (1984).

[12] A. M. Frieze, Discr. Math. **81**, 171 (1990).

[13] M. Weigt and A. K. Hartmann, *Phys. Rev. Lett.* **84**, 6118 (2000).

[14] M. Bauer and O. Golinelli, Eur. Phys. J. B **24**, 339 (2001).

[15] R. E. Tarjan and A. E. Trojanowski, SIAM J. Comp. **6**, 537 (1977).

[16] M. E. J. Newman und G. T. Barkema, *Monte Carlo Methods in Statistical Physics* (Clarendon Press, Oxford, 1999).

[17] D. P. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, (Cambridge University Press, Cambridge 2000).

[18] L. E. Reichl, *A modern Course in Statistical Physics*, (Wiley, New York 1998).

[19] M. Weigt and A. K. Hartmann, Europhys. Lett. **62**, 533 (2003).

[20] E. Marinari and G. Parisi, Europhys. Lett. **19**, 451 (1992).

[21] K. Hukushima and K. Nemoto, J. Phys. Soc. Jpn. **65**, 1604 (1996).

[22] W. Barthel and A. K. Hartmann, to appear in Phys. Rev. B (2004), preprint cond-mat/0403193.

[23] A. K. Hartmann and M. Weigt, J. Theor. Comp. Sci. **265**, 199 (2001).

[24] M. Weigt and A. K. Hartmann, Phys. Rev. E **63**, 056127 (2001).

[25] M. Weigt and A. K. Hartmann, Phys. Rev. Lett. **86**, 1658 (2001).

[26] Reviews on spin glasses can be found in: K. Binder and A. P. Young, Rev. Mod. Phys. **58**, 801 (1986); K. H. Fisher and J. A. Hertz, *Spin Glasses*, (Cambridge University Press, Cambridge 1991); A. P. Young (ed.), *Spin glasses and Random Fields*, (World Scientific, Singapore 1998).

[27] M. Mézard, G. Parisi, M. A. Virasoro, *Spin Glass Theory and Beyond*, (World Scientific, Singapore 1987).

[28] D. Sherrington and S. Kirkpatrick, Phys. Rev. Lett. **35**, 1792 (1975).

[29] G. Parisi, Phys. Rev. Lett. **43**, 1754 (1979); J. Phys. A **13**, 1101 (1980); **13**, 1887 (1980); **13**, L115 (1980); Phys. Rev. Lett. **50**, 1946 (1983).

[30] K. H. Fisher and J. A. Hertz, *Spin Glasses*, (Cambridge University Press, Cambridge 1991).

[31] R. Rammal, G. Toulouse, and M. A. Virasoro, Rev. Mod. Phys. **58**, 765 (1986).

[32] M. Talagrand, C.R.A.S. **337**, 111 (2003).

[33] A. P. Young, Phys. Rev. Lett. **51**, 13 (1983).

[34] G. Parisi, F. Ritort and F. Slanina, J. Phys. A **26**, 3775 (1993).

[35] A. Billoire, S. Franz, and E. Marinari, J. Phys. A **36** (2003).

[36] J. Sinova, G. Canright, and A. H. MacDonald, Phys. Rev. Lett. **85**, 2609 (2000).

[37] J. Sinova, G. Canright, H. E. Castillo, and A. H. MacDonald, Phys. Rev. B **63**, 104427 (2001).

[38] G. Hed, A. P. Young, and E. Domany, Phys. Rev. Lett. **92**, 157201 (2004).

[39] F. Krzakala and O. C. Martin, Phys. Rev. Lett. **85**, 3013 (2000).

[40] M. Palassini and A. P. Young, Phys Rev. Lett. **85**, 3017 (2000).

[41] R. Monasson and R. Zecchina, Phys. Rev. Lett. **76**, 3881 (1996); Phys. Rev. E **56**, 1357 (1997).

[42] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky, Nature **400**, 133 (1999).

[43] G. Biroli, R. Monasson, and M. Weigt, Eur. Phys. J. B **14**, 551 (2000).

[44] M. Mézard, G. Parisi, and R. Zecchina, Science **297**, 812 (2002).

[45] M. Mézard and R. Zecchina, Phys. Rev. E **66**, 056126 (2002).

[46] M. Weigt, in: A. K. Hartmann and H. Rieger (eds.), *New Optimization Algorithms in Physics*, 121, (Wiley-VCH, Weinheim 2004).

[47] R. Zecchina, in: A. K. Hartmann and H. Rieger (eds.), *New Optimization Algorithms in Physics*, 183, (Wiley-VCH, Weinheim 2004).

[48] A. K. Hartmann, J. Phys. A **33**, 657 (2000).

7 Statistical mechanics of vertex covers on a random graph

7.1 Introduction

In the last section, the vertex cover (VC) problem was investigated from a numerical point of view. After the introduction of some algorithms which are able to construct vertex covers on arbitrary graphs, the problem was investigated on random graphs.

The motivation was the following. Studying random graphs, one can infer properties of the typical-case solvability of typical-case algorithmic behavior for whole classes of graphs, as represented by the random graph ensemble $\mathcal{G}(N, c/N)$. In the last section, we have already seen some numerical evidence for the existence of a c-dependent phase transition. In the thermodynamic limit, the existence of a threshold function $x_{\mathrm{c}}(c)$ is conjectured. Given as $x > x_{\mathrm{c}}(c)$, a graph drawn from $\mathcal{G}(N, c/N)$ can almost surely be covered with at most xN vertices. Here, as in the introductory section on random graphs, "almost surely" means that the probability tends to one for $N \to \infty$. If, on the other hand, a value $x < x_{\mathrm{c}}(c)$ is given, there are almost surely no VCs with at most xN vertices. This behavior resembles very much phase transitions in statistical mechanics, and it is tempting to use tools borrowed from statistical mechanics to calculate the transition line $x_{\mathrm{c}}(c)$ or other statistical properties of minimum vertex covers on random graphs $G \in \mathcal{G}(N, c/N)$. This will be the task of this section. The presentation follows mainly [1, 2].

Another numerical observation which is even more interesting for computer scientists was the solution-time pattern. If one tries to answer the decision problem of whether a VC of at most xN vertices exists or not for some given graph, the answer can be easily found for sufficiently large x. The median solution time grows only linearly with N in this case. This behavior changes if we approach the phase boundary $x_{\mathrm{c}}(c)$ by decreasing the allowed VC size x. First, still inside the coverable phase, i. e., above $x_{\mathrm{c}}(c)$, exponential solution times set in. The maximal solution time is required exactly at the phase boundary, i. e., for $x \simeq x_{\mathrm{c}}(c)$. If we take $x < x_{\mathrm{c}}(c)$, almost surely no VC of the required size exists. The algorithmic proof of this fact still requires exponential times, but these are smaller than the one required at the phase boundary, it decreases monotonously with decreasing x. The problem instances which are hardest to solve are thus situated close to the phase boundary, they are said to be *critically constrained*. Analyzing this phenomenon for a simple backtracking algorithm will be a task within the next chapter.

At this point, a short historical remark is necessary. The use of statistical-mechanics methods in optimization goes back to the mid-eighties of the 20th century, where matching [3], graph partitioning [4] and TSP [5] where studied. A real interaction with computer science, and a strong increase in the activity in statistical physics was, however, begun only in the nineties by the discovery of the same type of phase transitions as discussed above in the satisfiability problem [6, 7], and the subsequent statistical-mechanics approach pioneered by [8]. These phenomena will be discussed in Chap. 10. An overview of the interdisciplinary evolution of the field can be seen in special editions [9–11] dedicated, by various journals, to this field.

7.2 The first-moment bound

Before locating the phase transition with statistical-mechanics tools, we give a short description of the so-called *first-moment method*. It allows us to derive a rigorous *lower bound* on $x_c(c)$. From the point of view of statistical mechanics, the first-moment method is equivalent to the annealed average mentioned before. Instead of averaging the logarithm of the partition function, we average directly the partition function and thus get a bound on the free energy.

For a moment we work in the ensemble $\mathcal{G}(N, M)$ where both the vertex number N and the edge number $M = \frac{c}{2}N$ are fixed. According to the discussion in Chap. 3, this should not change the properties of large graphs since the edge number is concentrated close to its average value also in the $\mathcal{G}(N, c/N)$-ensemble. Within the first-moment method, it gives, however, a bound which is better and also easier to derive.

The first-moment method is based on a simple bound on the probability that a graph drawn from $\mathcal{G}(N, \frac{c}{2}N)$ can be covered with $X = xN$ vertices. This probability is obviously bounded from above by the average number of vertex covers of cardinality $X = xN$ which a graph allows for:

$$\mathrm{Prob}\left(G \in \mathcal{G}(N, \frac{c}{2}N) \text{ has VC of size } xN\right) \leq \overline{\mathcal{N}_{vc}(G, xN)} \tag{7.1}$$

since a graph without a VC of the desired size does not contribute to either side, whereas a graph with VC contributes one on the left-hand side, and its number (≥ 1) of VCs on the right-hand side. The latter object is easily calculated. There are $\binom{N}{xN}$ vertex subsets of cardinality xN, i. e., potential vertex covers. In each of these sets, a pair of vertices being connected by an edge has, with probability $(1-x)^2$, both end-points uncovered – violating thus the VC constraint. With probability $1-(1-x)^2 = x(2-x)$, at least one end-vertex is covered and the VC constraint is satisfied. In a vertex cover all constraints have to be fulfilled simultaneously, we thus conclude

$$\overline{\mathcal{N}_{vc}(G, xN)} = \binom{N}{xN} \left[x(2-x)\right]^{\frac{c}{2}N} \ . \tag{7.2}$$

Using Stirling's formula to express the binomial coefficient (see Sec. 5.3), we find

$$\overline{\mathcal{N}_{vc}(G, xN)} = \exp\left\{N\left[-x\ln x - (1-x)\ln(1-x) + \frac{c}{2}\ln\{x(2-x)\}\right]\right\} \ . \tag{7.3}$$

For fixed c and large enough x, the argument of the exponential in this equation is positive, the expected number of VCs becomes exponentially large, and Eq. (7.1) is trivially fulfilled. At a certain $x_{an}(c)$, however, this argument changes sign, and the expected number approaches zero exponentially in the thermodynamic limit. According to Eq. (7.1), the graph G almost surely does not have a vertex cover of less than $x_{an}(c)N$ vertices, i. e.,

$$x_{an}(c) < x_c(c) \tag{7.4}$$

provides a lower bound on the actual location of the phase transition, with

$$0 = x_{an}(c)\ln x_{an}(c) - (1-x_{an}(c))\ln(1-x_{an}(c)) + \frac{c}{2}\ln\{x_{an}(c)[2-x_{an}(c)]\}\ . \tag{7.5}$$

For finite c, the last equation has to be solved numerically. For large average degree c, the asymptotically leading terms can, however, be extracted analytically. One finds

$$x_{an}(c) = 1 - 2\frac{\ln c}{c} + \mathcal{O}\left(\frac{\ln\ln c}{c}\right)\ , \tag{7.6}$$

as can be checked most easily by plugging this result into the equation for $x_{an}(c)$, and by expanding the logarithmic terms. It is expected, that this bound will become tight for large c, i. e., that the correct asymptotic behavior is reproduced by the last expression. More precisely, the asymptotic is given by [12]

$$x_c(c) = 1 - \frac{2}{c}(\ln c - \ln\ln c - \ln 2 + 1) + o\left(c^{-1}\right)\ . \tag{7.7}$$

7.3 The hard-core lattice gas

In order to go beyond the first-moment bound we have to use statistical-mechanics methods which are very similar to those used in Sec. 5.4, analyzing the thermodynamic properties of the ferromagnetic Ising model on a random graph. To do this, we first have to bijectively map the combinatorial problem to a statistical physics one, establishing in a sense a dictionary between the original problem and the tool box which we want to use.

In Sec. 6.6, we have already seen that VC can be reinterpreted as a model of non-overlapping (i. e., hard) particles of radius one on a graph. Any subset $U \subset V$ of the vertex set can be encoded by a configuration of N binary variables:

$$\nu_i := \begin{cases} 0 & \text{if} \quad i \in U \\ 1 & \text{if} \quad i \notin U\,. \end{cases} \tag{7.8}$$

Why we choose to set ν_i to *zero* for vertices in U, and not to one, becomes clear if we look at the vertex cover constraint. An edge is covered by the elements in U iff at most one of the two end-points has $\nu = 1$. So the variables ν_i can be interpreted as occupation numbers of vertices

by the center of a particle. The covering constraint translates into a hard sphere constraint. If a vertex is occupied, i. e., $\nu_i = 1$, then all neighboring vertices have to be empty. We are thus led to particles of *chemical radius* one, i. e., two particles have to have at least distance two on the graph. The allowed configurations of particles are exactly the independent sets defined in Sec. 3.1.4. The constraint can be enforced by the indicator function (6.14),

$$\chi(\nu_1, \ldots, \nu_N) = \prod_{i<j} (1 - J_{ij}\nu_i\nu_j) \tag{7.9}$$

with (J_{ij}) being the adjacency matrix with entries $J_{ij} = 1$ if $\{i, j\} \in E$, and zero otherwise. The function χ takes value one whenever $\underline{\nu} = (\nu_1, \ldots, \nu_N)$ corresponds to an allowed particle packing (or a VC) because all factors equal one. Whenever there is an uncovered edge, i. e., $J_{ij} = \nu_i = \nu_j = 1$, the value of χ equals zero. Having this in mind, we write down the grand-canonical partition function of the hard-sphere lattice gas

$$\Xi = \sum_{\{\nu_i=0,1\}} \exp\left(\mu \sum_i \nu_i\right) \chi(\underline{\nu}) \tag{7.10}$$

where μ is a chemical potential which can be used to control the particle number, or the cardinality of U, because it gives different weights to allowed packings of different particle numbers. For regular lattices, this model is well studied as a lattice model for the fluid–solid transition.

Denoting the grand canonical average of a function f as

$$\langle f(\underline{\nu})\rangle_\mu = \Xi^{-1} \sum_{\{\nu_i=0,1\}} \exp\left(\mu \sum_i \nu_i\right) \chi(\underline{\nu})\, f(\underline{\nu}) \tag{7.11}$$

we can calculate the average occupation density

$$\rho(\mu) = \frac{1}{N}\left\langle \sum_i \nu_i \right\rangle_\mu = \frac{\partial}{\partial\mu}\frac{\ln \Xi}{N}\,. \tag{7.12}$$

Minimum vertex covers correspond to densest particle packings. Considering the weights in (7.10), it becomes obvious that the density $\rho(\mu)$ is an increasing function of the chemical potential. Densest packings, or minimum vertex covers, are thus obtained in the limit $\mu \to \infty$:

$$x_{\mathrm{c}}(c) = 1 - \lim_{\mu\to\infty} \nu(\mu)\,. \tag{7.13}$$

The thermodynamic limit $N \to \infty$ is implicitly assumed in the last expression.

7.4 Replica approach

Now we calculate the disorder-averaged logarithm of the partition function (7.10) using the replica approach. We follow closely the treatment of the disordered ferromagnet in Sec. 5.3, hence the reader should be familiar with that section before continuing here.

7.4.1 The replicated partition function

The main problem in handling the grand-canonical partition function (7.10) is caused by the disorder due to the random structure of the underlying graph, i. e., of the edge set E. To calculate typical properties we therefore have to evaluate the disorder average of $\ln \Xi$ over the random graph ensemble. This can again be achieved by the replica trick,

$$\overline{\ln \Xi} = \lim_{n \to 0} \frac{\overline{\Xi^n} - 1}{n} \tag{7.14}$$

where the over-bar denotes the disorder average over the random-graph ensemble. Taking n to be a positive integer at the beginning, we may replace the original system by n identical copies (including identical edge sets). In this case, the disorder average is easily obtained, and the $n \to 0$ limit has to be achieved later by analytically continuing in n. We may thus write, where n is a natural number,

$$\overline{\Xi^n} = \sum_{\{\nu_i^a\}} \exp\left(\mu \sum_{i,a} \nu_i^a\right) \overline{\prod_{i<j} \prod_{a=1}^{n} (1 - J_{ij} \nu_i^a \nu_j^a)} \,. \tag{7.15}$$

Here a denotes the replica index which runs from 1 to n. Putting edges independently with probability c/N, i. e., one has the contribution 1 with probability $1 - c/N$ and the contribution $(1 - \nu_i^a \nu_j^a)$ with probability c/N, results in

$$\begin{aligned} \overline{\Xi^n} &= \sum_{\{\nu_i^a\}} \exp\left(\mu \sum_{i,a} \nu_i^a\right) \prod_{i<j} \left[1 - \frac{c}{N} + \frac{c}{N} \prod_a (1 - \nu_i^a \nu_j^a)\right] \\ &= \sum_{\{\nu_i^a\}} \exp\left(\mu \sum_{i,a} \nu_i^a - \frac{cN}{2} + \frac{c}{2N} \sum_{i,j} \prod_a (1 - \nu_i^a \nu_j^a) + \mathcal{O}(1)\right) . \end{aligned} \tag{7.16}$$

We used $\log(1 - x) = -x + \mathcal{O}(x^2)$ and $\sum_{i<j} = \frac{1}{2} \sum_{i,j} + \mathcal{O}(N)$ to arrive at the second line. Following the ideas presented in the analysis of the Ising model on a random graph, we introduce 2^n order parameters

$$c(\vec{\xi}) = \frac{1}{N} \sum_i \prod_a \delta_{\xi^a, \nu_i^a} \tag{7.17}$$

as the fraction of vertices showing the replicated occupation variable $\vec{\xi} \in \{0, 1\}^n$. The exponent in the last line of Eq. (7.16) depends only on this quantity, for the different contributions

appearing there we have:

$$\begin{aligned}\sum_{i,a}\nu_i^a &= \sum_i\sum_{\vec{\xi}}\left(\prod_a\delta_{\xi^a,\nu_i^a}\sum_a\xi^a\right)\\ &= N\sum_{\vec{\xi}}c(\vec{\xi})\sum_a\xi^a \qquad (7.18)\\ \frac{c}{N}\sum_{i<j}\prod_a(1-\nu_i^a\nu_j^a) &= \frac{c}{2N}\sum_{i,j}\prod_a(1-\nu_i^a\nu_j^a)-\mathcal{O}(N^0)\\ &= \frac{c}{2N}\sum_{i,j}\sum_{\vec{\xi}}\left(\prod_a\delta_{\xi^a,\nu_i^a}\sum_{\vec{\zeta}}\left(\prod_a\delta_{\zeta^a,\nu_j^a}\prod_a(1-\xi^a\zeta^a)\right)\right)\\ &= \frac{cN}{2}\sum_{\vec{\xi},\vec{\zeta}}c(\vec{\xi})c(\vec{\zeta})\prod_a(1-\xi^a\zeta^a) \qquad (7.19)\end{aligned}$$

This can be plugged into Eq. (7.16), and the values of the order parameters $c(\vec{\xi})$ can be forced by a product of Kronecker symbols,

$$\begin{aligned}\overline{\Xi^n} = &\sum_{\{c(\vec{\xi})\}}\sum_{\{\nu_i^a\}}\prod_{\vec{\xi}}\delta\left(c(\vec{\xi}),\frac{1}{N}\sum_i\prod_a\delta_{\xi^a,\nu_i^a}\right)\\ &\times\exp\left\{N\left(-\frac{c}{2}+\mu\sum_{\vec{\xi},a}c(\vec{\xi})\xi^a+\frac{c}{2}\sum_{\vec{\xi},\vec{\zeta}}c(\vec{\xi})c(\vec{\zeta})\prod_a(1-\xi^a\zeta^a)\right)\right\} \qquad (7.20)\end{aligned}$$

where $\delta(\cdot,\cdot)$ also denotes the Kronecker symbol; for readability we did not use the usual notation. The dependence on the microscopic configurations $\{\nu_i^a\}$ is completely contained in this δ, so the summation over the microscopic configurations counts the number of possible realizations of the order parameter. This means that, for a given order parameter $c(\vec{\xi})$, defined by 2^n values, we want to know how many configurations there are, where the number of replicated vertices having value $(0,\ldots,0,0)\in\{0,1\}^n$ is $Nc((0,\ldots,0,0))$, *and* the number of replicated vertices having value $(0,\ldots,0,1)$ is $Nc((0,\ldots,0,1))$, …, *and* the number of replicated vertices having value $(1,\ldots,1,1)$ is $Nc((1,\ldots,1,1))$. Consequently the number is given by the multinomial coefficient $N!/\prod_{\vec{\xi}}(c(\vec{\xi})N)!$ which can be simplified using Stirling's formula and using $\sum_{\vec{\xi}}c(\vec{\xi})=1$. For large N, the sum over the order parameters $c(\vec{\xi})\in\{0,1/N,2/N,\ldots,1\}$ can be replaced by an integration. We thus get

$$\begin{aligned}\overline{\Xi^n} = \int\mathcal{D}c(\vec{\xi})\exp\Bigg\{N\Bigg(&-\sum_{\vec{\xi}}c(\vec{\xi})\ln c(\vec{\xi})-\frac{c}{2}+\mu\sum_{\vec{\xi},a}c(\vec{\xi})\xi^a\\ &+\frac{c}{2}\sum_{\vec{\xi},\vec{\zeta}}c(\vec{\xi})c(\vec{\zeta})\prod_a(1-\xi^a\zeta^a)\Bigg)\Bigg\} \qquad (7.21)\end{aligned}$$

where the integration runs over all normalized distributions $c(\vec{\xi})$, i. e., $\sum_{\vec{\xi}} c(\vec{\xi}) = 1$. In the large-$N$ limit, the integration can be solved by the saddle-point method. The saddle-point equation can be obtained by variation of the exponent in (7.21) with respect to all allowed $c(\vec{\xi})$. This means we seek for a saddle point for all possible $\vec{\xi}'$ simultaneously, i. e., we calculate the derivative $\partial / \partial c(\vec{\xi}')$ of the exponent and set it to zero:

$$c(\vec{\xi}) = \exp\left\{ -1 - \lambda + \mu \sum_a \xi^a + c \sum_{\vec{\zeta}} c(\vec{\zeta}) \prod_a (1 - \xi^a \zeta^a) \right\} . \tag{7.22}$$

λ is a Lagrange multiplier introduced in order to guarantee the normalization of $c(\vec{\xi})$. For $n \to 0$, it will tend to $\lambda = c + 1$. Before we can, however, calculate this limit, we have to introduce some ansatz for $c(\vec{\xi})$ as even the dimensionality of $c(\vec{\xi})$ still depends on n. In the next subsection, we are going to use the simplest-possible, i. e., the replica-symmetric ansatz. As this ansatz is found to be valid only for a finite connectivity range, we also include some intuitive discussion on replica-symmetry breaking, two subsections later.

7.4.2 Replica-symmetric solution

As we have already explained, we are now using the so-called replica-symmetric ansatz, which in our case assumes that the order parameter $c(\vec{\xi})$ depends on $\vec{\xi}$ only via $\sum_a \xi_a$. In this case we are able to write, as in Eq. (5.44),

$$c(\vec{\xi}) = \int dh \, P(h) \frac{\exp(\mu h \sum_a \xi_a)}{(1 + e^{\mu h})^n} \tag{7.23}$$

where $P(h)$ is the probability distribution of the effective fields.

The role of effective fields becomes clear if we go back for a moment to the original disordered and unreplicated model. The grand-canonical probability for a microscopic configuration $(\nu_1, \ldots, \nu_N) \in \{0, 1\}^N$ is given by

$$P(\nu_1, \ldots, \nu_N) = \frac{1}{\Xi} e^{\mu \sum_i \nu_i} \prod_{i<j} (1 - J_{ij} \nu_i \nu_j) . \tag{7.24}$$

The marginal probability for the single vertex i can be calculated by tracing over all other microscopic degrees of freedom,

$$P^{(i)}(\nu_i) = \sum_{\nu_1} \cdots \sum_{\nu_{i-1}} \sum_{\nu_{i+1}} \cdots \sum_{\nu_N} P(\nu_1, \ldots, \nu_N) . \tag{7.25}$$

Due to the binary character of ν_i this can be written as

$$P^{(i)}(\nu_i) = \frac{e^{\mu h_i \nu_i}}{1 + e^{\mu h_i}} . \tag{7.26}$$

The denominator ensures the correct normalization, whereas the effective field h_i is given by

$$h_i = \frac{1}{\mu} \ln \frac{P^{(i)}(1)}{P^{(i)}(0)} \,. \tag{7.27}$$

If we only look to this single site, we are not able to say whether it is an isolated site exposed to an exterior field h_i, or if it is a part of a complicated interacting system. In this sense, the effective field h_i incorporates the effect of all other vertices on site i. The distribution $P(h)$ is simply the histogram of these fields,

$$P(h) = \frac{1}{N} \sum_i \delta(h - h_i) \,. \tag{7.28}$$

After this interpretation of effective fields, we return to the saddle-point equation (7.22), and we plug in ansatz (7.23). This allows us to eliminate the $\vec{\xi}$-dependencies and, in the replica limit $n \to 0$, to go to a non-linear integral equation for $P(h)$. First we introduce the abbreviation $y = \mu \sum_a \xi^a$, then the left-hand side of Eq. (7.22) reads

$$c(\vec{\xi}) \overset{n\to 0}{\to} \int dh\; P(h)\, e^{hy} \,. \tag{7.29}$$

The right-hand side is a bit more involved. The order-parameter dependent term is

$$\begin{aligned}
\sum_{\vec{\zeta}} c(\vec{\zeta}) \prod_a (1 - \xi^a \zeta^a) \quad &= \quad \int dh\; \frac{P(h)}{(1+e^{\mu h})^n} \sum_{\vec{\zeta}} e^{\mu h \sum_a \zeta^a} \prod_a (1 - \xi^a \zeta^a) \\
&= \quad \int dh\; \frac{P(h)}{(1+e^{\mu h})^n} \prod_a \left[\sum_{\zeta^a} e^{\mu h \zeta^a} (1 - \xi^a \zeta^a) \right] \\
&= \quad \int dh\; \frac{P(h)}{(1+e^{\mu h})^n} (1 + e^{\mu h})^{n - y/\mu} \\
&\overset{n\to 0}{\to} \quad \int dh\; P(h)\; (1 + e^{\mu h})^{-y/\mu} \,.
\end{aligned} \tag{7.30}$$

The last but one step results from the observation that the terms in the product over the replicas depend only on whether $\xi^a = 0$ ($n - y/\mu$ times) or $\xi^a = 1$ (y/μ times). The saddle-point equation (7.22) thus reads in the limit $n \to 0$

$$\int dh\; P(h) e^{hy} = \exp\left\{ -c + y + c \int dh\; P(h) (1 + e^{\mu h})^{-y/\mu} \right\} \tag{7.31}$$

and has to be valid for arbitrary y. It can be rewritten as

$$P(h) = \sum_{d=0}^{\infty} e^{-c} \frac{c^d}{d!} \int dh_1 \cdots dh_d\; P(h_1) \cdots P(h_d)\; \delta\left(h - 1 + \frac{1}{\mu} \sum_{i=1}^{d} \ln(1 + e^{\mu h_i}) \right) \tag{7.32}$$

as can be verified easily by plugging this equation into the left-hand site of Eq. (7.31).

For finite chemical potentials μ, this equation cannot be solved analytically, one has to use, e. g., the population dynamical algorithm discussed before. We are, however, interested in particle packings of maximal density, or equivalently minimum VCs, which correspond to the limit $\mu \to \infty$. The chemical potential μ appears in Eq. (7.32) only inside the delta function, but there the limit is easily calculated:

$$\lim_{\mu\to\infty} \frac{1}{\mu} \ln(1 + e^{\mu h_i}) = \max(0, h_i) \tag{7.33}$$

since the term $e^{\mu h_i}$ becomes exponentially small (large) compared to 1 for $h_i < 0$ ($h_i > 0$). We thus find in this limit

$$P(h) = \sum_{d=0}^{\infty} e^{-c} \frac{c^d}{d!} \int dh_1 \cdots dh_d \, P(h_1) \cdots P(h_d) \, \delta \left(h - 1 + \sum_{i=1}^{d} \max(0, h_i) \right). \tag{7.34}$$

Since $\max(0, h_i)$ is non-negative, the field h cannot be larger than 1. If we further assume the fields to take only integer values (as can be physically motivated on the basis of energy differences (cf. the corresponding discussion for the Ising model in Sec. 5.4.3), we are led to the ansatz

$$P(h) = \sum_{\ell=-\infty}^{1} r_\ell \delta(h - \ell), \tag{7.35}$$

where the factor r_ℓ gives the probability that the effective field of a randomly selected vertex equals ℓ.

Under this ansatz, equation (7.34) can be easily solved. First we consider field $h = 1$ on both sides. On the right-hand side, it can be generated if and only if all fields h_i are smaller or equal to zero, $h_i \leq 0$, because then the max functions vanish. For each field, this happens with probability $1 - r_1$. Comparing the coefficients of $\delta(h - 1)$ on both sides, we find

$$\begin{aligned} r_1 &= \sum_{d=1}^{\infty} e^{-c} \frac{c^d}{d!} (1 - r_1)^d \\ &= e^{-c r_1} \, . \end{aligned} \tag{7.36}$$

This equation can be solved explicitly using the real branch of the Lambert-W function, which is defined by the equation

$$W(x) = x e^{-W(x)} \, , \tag{7.37}$$

hence

$$r_1 = \frac{W(c)}{c} \, . \tag{7.38}$$

To obtain r_0, the factors of $\delta(h)$ on the right-hand side of Eq. (7.34) can be calculated similarly. In this case, exactly one of the fields $h_1, \ldots, h_d$ has to be one, all others again smaller or equal

to zero. Taking into account that there are d possibilities for selecting which field is positive, we get

$$\begin{aligned} r_0 &= \sum_{d=1}^{\infty} e^{-c} \frac{c^d}{d!} r_1 (1-r_1)^{d-1} d \\ &= c r_1 e^{-c r_1} \\ &= \frac{W(c)^2}{c} \end{aligned} \tag{7.39}$$

where we have plugged in solution (7.38) in the last line. Now we are finally able to calculate the fraction $x_{\mathrm{c}}(c)$ of vertices belonging to minimum VCs. We use the fact that, for positive effective fields, the vertices are occupied with probability one (for $\mu = \infty$), while for negative effective fields, they are never occupied. The case of vanishing fields $h = 0$ is slightly more subtle, because the limit μh could be non-zero and give rise to a non-trivial average occupation number. However, a more subtle analysis of the limit $\mu \to \infty$ shows that the corresponding vertices have, on average, occupation 1/2 [2]. This argument will be given later on in the context of the belief-propagation algorithm in Chap. 9. The highest possible particle density is thus given by

$$\rho(\mu \to \infty) = r_1 + \frac{1}{2} r_0 = \frac{2W(c) + W(c)^2}{2c} \,. \tag{7.40}$$

Due to the discussed correspondence of empty vertices in particle packings and elements of vertex covers, we conclude that minimum vertex covers on large random graphs contain a fraction

$$x_{\mathrm{c}}(c) = 1 - \frac{2W(c) + W(c)^2}{2c} \tag{7.41}$$

of all vertices. We have thus found the minimum vertex cover size using the tools of statistical mechanics.

In the last chapter, the so-called backbone a was also discussed, and an algorithm for its determination was introduced. The backbone contains all those vertices i which assume the same value ν_i in all minimum VCs, i. e., which are either always covered or which are never covered. These two cases allow us to distinguish two different backbone types, namely B^{ac} for the always covered, and B^{auc} for the always uncovered vertices. Going back to the discussion in the last paragraph, we can determine the backbones from the local effective fields,

$$\begin{aligned} B^{ac} &= \{i \in V;\ h_i < 0\} \\ B^{auc} &= \{i \in V;\ h_i = 1\} \end{aligned} \tag{7.42}$$

which leads to the backbone sizes

$$\begin{aligned} b^{ac}(c) = \frac{|B^{ac}|}{N} &= 1 - r_1 - r_0 \\ b^{auc}(c) = \frac{|B^{auc}|}{N} &= r_1 \,, \end{aligned} \tag{7.43}$$

and the total backbone is the union of both partial backbones.

In Figs 6.14 and 6.24 of Chap. 6, these results are compared with numerical data. One can see that, for relatively small average vertex degrees c, the coincidence is perfect, whereas for larger c there are some systematic deviations. The numerically predicted minimum VC sizes are larger than the numerical ones.

7.4.3 Beyond replica symmetry

To explain this discrepancy, we have to recall the assumptions made during our calculations:

- First, we assumed that the order parameter is replica symmetric. This means that $c(\vec{\xi})$ depends only on $\sum_a \xi^a$. This assumption allowed us to introduce the distribution of effective fields $P(h)$ as the order parameter which is relevant in the replica limit $n \to 0$.
- On top of this assumption, we used an ansatz containing only integer effective fields. As in the Ising model discussed in Section 5.4.3, this ansatz can be justified by looking to possible energy differences.

However, the last argument is not a rigorous one, so one can be tempted to relax this assumption. Looking to the saddle-point equation, we observe that it is also closed if we use instead fields which are Kth fractions of integers, for any arbitrary K. Having in mind that fields cannot be larger than one, the corresponding ansatz reads

$$P^{(K)}(h) = \sum_{\ell=-\infty}^{K} r_\ell^{(K)} \delta(h - \ell/K) \,. \tag{7.44}$$

The coefficients give the probabilities that the corresponding fields appear for a randomly selected vertex, in particular they have to be non-negative. Plugging this ansatz into the saddle-point equation, we can observe that this positivity constraint for non-integer fields is met only if the average degree c is larger than the Eulerian constant, $c > e \simeq 2.718$. In this case, thresholds $x_{\mathrm{c}}(c)$ are predicted which grow monotonously with K, but even asymptotically they remain smaller than the numerical prediction.

The fact that they are closer to the true threshold is, however, a hint that the inclusion of non-integer fields is not fundamentally wrong. Looking at the argument for integer fields, which contained the discussion of possible energy differences by flipping single spins, we did not take into account that effective fields arrive once we integrate out $N - 1$ degrees of freedom, i. e., by a complicated averaging procedure. In this sense, non-integer fields result from this complicated averaging procedure, and they are a sign of something more fundamental going on: for a spontaneously broken replica symmetry.

When we discussed the breaking of the spin-flip symmetry in ferromagnetic models, we saw that the space of realizable configurations broke into two disconnected clusters, one having positive, the other negative magnetization. The behavior is more drastic in the case of replica symmetry breaking, the number of clusters becomes very large (divergent in N). This means that, for $c < e$, there is only one large cluster of minimum vertex covers in the configuration space $\{0, 1\}^N$. Starting from one solution, one can reach every other by a path using only

minimum VCs which are, one after the other, separated by a finite number of bits. For higher graph degrees, i. e., for $c > e$, this is no longer true. There are many clusters which have macroscopic distances from each other. There, for every vertex $i \in \{1, \ldots, N\}$ and every cluster $\alpha \in \{1, \ldots, \mathcal{N}_{cl}\}$, one can define an effective field h_i^α. A simple field h_i for each vertex is no longer sufficient, and the simple replica symmetric ansatz is not able to capture the most important features of the true effective fields. Technically, the calculation is much more involved than the replica symmetric one [2], and a complete solution has not yet been found.

Zhou [13] has derived the same results using the cavity approach, for the replica-symmetry broken case he was even able to go beyond the results of the replica calculation. Parts of this approach will be presented in Chap. 9 in an algorithmic interpretation.

Bibliography

[1] M. Weigt and A. K. Hartmann, Phys. Rev. Lett **84**, 6118 (2000).

[2] M. Weigt and A. K. Hartmann, Phys. Rev. E **63**, 056127 (2001).

[3] M. Mézard and G. Parisi, J. Phys. Lett. **46**, L771 (1985).

[4] Y. Fu and P. W. Anderson, J. Phys. A **19**, 1605 (1986).

[5] M. Mézard and G. Parisi, J. Physique **47**, 1285 (1986).

[6] D. Mitchell, B. Selman, and H. Levesque, in: *Proc. of the 10th Natl. Conf. on Artificial Intelligence AAAI-92*, 459 (1992).

[7] B. Selman and S. Kirkpatrick, Science (Washington DC) **264**, 1297 (1994).

[8] R. Monasson and R. Zecchina, Phys. Rev. Lett. **76**, 3881 (1996); Phys. Rev **E56**, 1357 (1997).

[9] T. Hogg, B. A. Huberman, and C. Williams (eds.), Special issue of Artif. Intell. **81** (1996).

[10] O. Dubois, R. Monasson, B. Selman, and R. Zecchina (eds.), Special issue of Theor. Comp. Sci. **265** (2001).

[11] E. Marinari, H. Nishimori, and F. Ricci-Tersenghi (eds.), Special issue of J. Phys. A **36**, nr. 43 (2003).

[12] A. M. Frieze, Discr. Math. **81**, 171 (1990).

[13] H. Zhou, Eur. Phys. J. B **32**, 265 (2003).

8 The dynamics of vertex-cover algorithms

In the last chapter, we discussed the statistical properties of minimum vertex covers using statistical-mechanics tools. In this section we will, in addition, show how to characterize analytically the dynamical behavior of some of the algorithms which have already been introduced in Chap. 6, and which were used there in order to generate minimum vertex covers numerically. We will concentrate on three types of algorithms. The first example is a *complete branch-and-bound algorithm* which either constructs vertex covers of a given size, or proves their non-existence. In the second example, we analyze a class of *linear-time heuristic algorithms* which contain the already introduced leaf-removal procedure. These algorithms construct small, but not necessarily minimum vertex covers. As in the last chapter, we are interested in the typical-case behavior, i. e., in the most probable algorithmic dynamics on randomized problem instances. In the third and last example we study deviations from this average dynamics. Extensive deviations are exponentially improbable, but they can be exploited in *random-restart algorithms*. Also these algorithms are not complete in the sense that they are not able to prove that the constructed VC is minimum. Compared to branch-and-bound they may, in any case, cause an almost sure exponential speedup, with an exponentially small failure probability.

There are many other types of algorithms which we do not analyze here. A particularly important class are so-called stochastic local search (SLS) algorithms. They share the following idea. The algorithm starts with some trial solution, which in general will be far from optimal. Then, the algorithm tries to improve this configuration in a stochastic way, by randomly selecting variables and changing them according to local considerations. The, possibly, most famous example of this class of algorithm is simulated annealing [1], which has been introduced in Chap. 6. A second example is the WalkSAT algorithm [2] which is designed specifically for constraint satisfaction problems, see the discussion in Chap. 10.

Note that the behavior of the algorithms discussed here is intrinsically of non-equilibrium character, as in every algorithmic step the state of the system changes. To characterize the statistical properties of all minimum vertex covers, we have used equilibrium methods as, e. g., calculating the partition function and global order parameters. For the dynamics we have instead to apply methods which are much more related to non-equilibrium phenomena in physics as, e. g., stochastic growth processes or thermally activated transitions.

There is also one class of algorithm which is more related to equilibrium statistical-mechanics calculations, namely message-passing (or statistical inference) algorithms. These will be discussed in Chap. 9.

8.1 The typical-case solution time of a complete algorithm

Let us start the analytical description of algorithmic behavior with the case of a branch-and-bound algorithm, cf. [3]. The algorithm follows the lines of the analogous procedure introduced in Sec. 6.3, but it is very simplified. The reason is the following. Whereas the algorithm in Sec. 6.3 is designed to be as efficient as possible, here we are interested in an analytically tractable algorithm which, however, maintains the qualitative features of more involved implementations.

The algorithm is designed for the *vertex-cover decision problem*. Is it possible to find a vertex cover of a given graph which contains at most $X = xN$ vertices, with $0 < x < 1$? The algorithm thus does not solve directly the *minimum vertex-cover problem*, but a simple extension of the algorithm also solves the latter one. We run the algorithm first with $X = N - 1$, where vertex covers for sure exist, and then decrease X in steps of one until no VCs are found any longer. The smallest vertex cover found in this way is consequently a minimum one, and the total running time picks up, at most, a factor which is linear in N compared with that of the decision algorithm.

The analysis shown below follows [3], and is based on a similar approach to 3-SAT given in [4]. The approach of the latter paper was refined in subsequent publications [5].

8.1.1 The algorithm

Let us start with the definition of the algorithm itself. Given a graph $G = (V, E)$ of order $|V| = N$, we want to answer the question whether it has a VC V_{vc}, which has, at most, the cardinality $X = xN$.

As each vertex can be either covered or uncovered, there are $\binom{N}{X}$ possible configurations which can be arranged as the leaves of a binary configuration tree, which is traversed by the backtracking algorithm. The basic idea is to traverse the whole tree to search for vertex covers. At first we explain how the tree-traversal is organized (*branch*) then we show how much computational time can be saved by excluding subtrees where no covers can definitely be found (*bound*).

We introduce three states of vertices: *free*, *covered* or *uncovered*. The algorithm starts at the root of the tree where all vertices are *free*. The algorithm descends into the tree by selecting a *free* vertex i at random. Each such vertex has two subtrees corresponding to a covered/uncovered i. If i has neighboring vertices which are either *free* or *uncovered*, we mark i *covered* first. If the number of covered vertices does not exceed xN the descent continues. If the algorithm returns to i by backtracking, vertex i is set *uncovered* and the algorithm descends into the other subtree. If i has only covered neighbors, the order in which the two subtrees are visited is exchanged. Note that in this case, to find a solution, it is necessary to include the second subtree, where i is *covered*, but we include this case to facilitate the analysis.

The algorithm stops either if it has covered all edges before having used all covering marks (output: graph coverable) or if its has exhausted all covering marks in the last branch without having covered all edges (output: graph uncoverable).

The performance of this algorithm can be easily improved by introducing a *bound*. If, at any node, one of the subtrees can be proved to contain no VC of the desired maximal size, the corresponding subtree can be omitted. The bound used here is extremely simple. It forbids the marking of a vertex as uncovered if any of its neighbors is already marked uncovered. Otherwise some edges would remain uncovered. The algorithm is summarized below, where $G = (V, E)$ denotes the graph, $m(i) \in \{$*free,cov,uncov*$\}$ contains the mark of vertex i, and X equals the currently available number of marks. Initially we set $m(i) = free$ for all $i \in V$, and $X = xN$.

procedure vertex-cover(G, m, X)
begin
  if all edges are covered **then**
    stop;
  if $X = 0$ **then**
    return;
  Select a vertex i with ($m(i) =$ *free*) randomly;
  if i has neighbors j with $m(j) \neq$ *cov* **then**
  begin
    $m(i) \leftarrow$ *cov*;
    vertex-cover($G, m, X - 1$);
    if i has no neighbors with $m(j)$=*uncov* **then**
    begin
      $m(i) \leftarrow$ *uncov*;
      vertex-cover(G, m, X);
    end
  end
  else (all neighbors j of i have $m(j) =$ *cov*)
  begin
    $m(i) \leftarrow$ *uncov*;
    vertex-cover(G, m, X);
    $m(i) \leftarrow$ *cov*;
    vertex-cover($G, m, X - 1$);
  end
end

This algorithm is complete, i. e., it decides whether or not a graph is coverable with the desired number of covering marks. Due to backtracking it will, in general, need exponential time in order to decide this question. In the following, the solution time is measured in a computer- and implementation-independent fashion as the *number of visited nodes* in the configuration tree.

8.1.2 Some numerical observations

In Fig. 8.1, some numerical results for the typical solution time are given for $c = 2$. The behavior is qualitatively similar for other average degrees. One can clearly distinguish three different phases. A first phase A, which is present for high values of x, shows a linear typical solution time. This is remarkable since also in this parameter region every possible graph having N vertices can be generated with non-zero probability, i. e., also the worst cases, which are expected to require exponential solution time. These bad cases are, however, extremely rare and do not influence the average behavior. The simple heuristic introduced above is good enough to provide a vertex cover in the very first descent into the configuration tree. A cartoon of the visited part of this tree is shown in Fig. 8.2, it is (almost) linear and thus contains only a linear number of nodes.

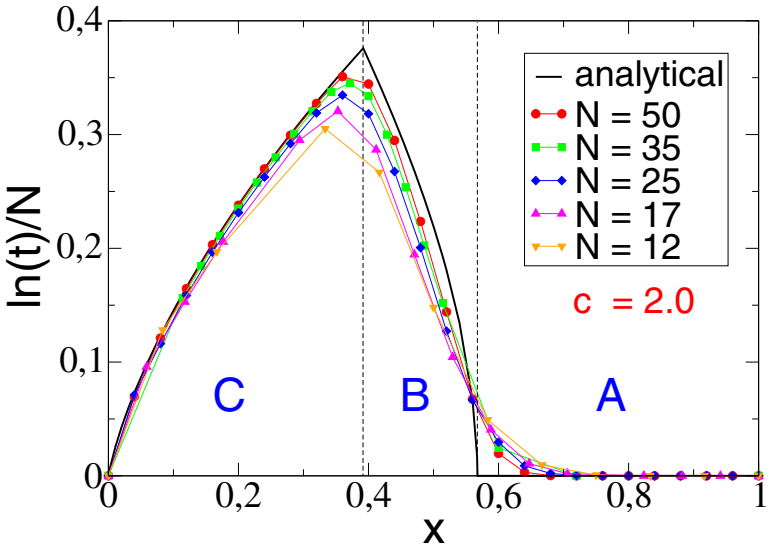


Figure 8.1: Normalized and averaged logarithm of running time of the algorithm as a function of the fraction x of coverable vertices. The solid line is the result of the annealed calculation. The symbols represent the numerical data for $N = 12, 17, 25, 35, 50$; lines are a guide to the eye only. One can clearly distinguish three dynamical phases, a linear (A) and two exponential ones (B and C), the first being coverable, the second uncoverable. The plot shows data for $c = 2.0$; graphs of other average degrees show the same qualitative behavior, with the complexity peak shifted according to the changed VC threshold $x_c(c)$.

Still inside the coverable phase, a sharp onset of exponential time complexity appears, note that the logarithm of the solution time is displayed. In phase B, which is situated between this dynamical transition and the coverable–uncoverable transition, the graph still has VCs of the desired cardinality, but these are hard to find for the heuristic. The corresponding shape of the

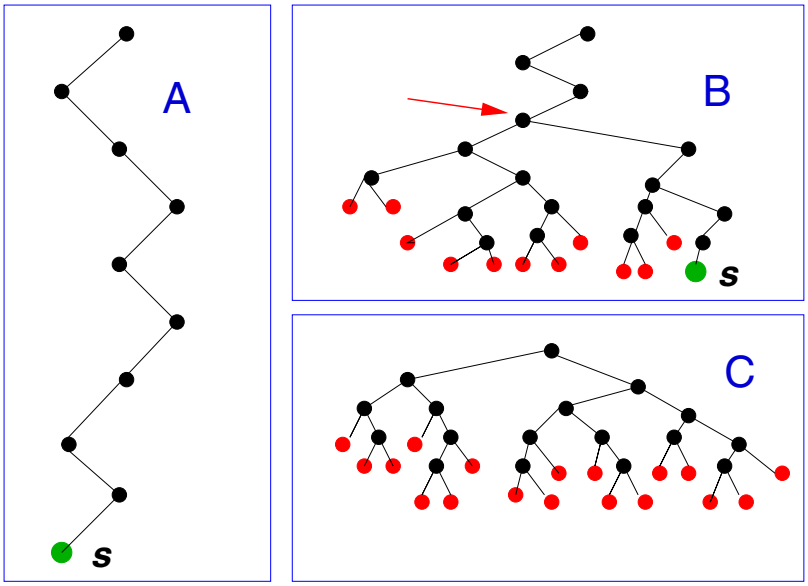


Figure 8.2: Typical shape of the configuration tree in the three dynamical phases, with black dots denoting internal nodes and gray ones, leaves. Solutions are marked by "S", other leaves correspond to contradictions. In the easy and coverable phase A, the algorithm is successful already in its first descent, no (or nearly no) backtracking is necessary. In the hard coverable phase B, the algorithm has to backtrack. The solution time is dominated by the number of steps the algorithm has to make before leaving the first subtree without solutions. This point is marked by the arrow. In the hard and uncoverable phase C, the algorithm proves the non-existence of a VC of the desired size after an exponential number of algorithmic steps.

visited configuration tree is again given in Fig. 8.2. At the node marked by the arrow, a "bad" choice was made. The search entered into a subtree which does not contain *any* solution, in the figure it is the left subtree. The algorithm escapes from this subtree by exponentially many backtracks which dominate the total solution time.

The solution time grows if we approach the static transition point $x_c(c)$. There, the system enters the third phase, which is characterized by uncoverable instances (with the available number of covering marks). The proof of this uncoverability takes an exponentially long time, but it becomes more efficient if we lower x and move away from the transition point. We thus observe that the hardest instances are, in fact, located at the coverable–uncoverable phase transition.

8.1.3 The first descent into the tree

The analysis of the first descent into the configuration tree is straightforward for our algorithm, as it forms a Markov process of random graphs. In every time step, one vertex and all its

incident edges are covered and can be regarded as removed from the graph. As the order of appearance of the vertices is not correlated to its geometrical structure, the graph remains a random graph. Now the order and the average degree depend on the time. After T steps, we consequently find a graph from the ensemble $\mathcal{G}(N-T, c/N)$ having $N(T) = N-T$ vertices. As the edge probability p remains unchanged $p = c/N$, i. e., *not* $c/N(T)$, the average vertex degree decreases from c to $c(T) \simeq N(T)p = (1-T/N)c$.

For large N, it is reasonable to work with the *rescaled time* $t = T/N$, which becomes continuous in the thermodynamic limit. In this notation, our generated graph is in $\mathcal{G}((1-t)N, c/N)$.

At time t, an isolated vertex is now found with probability $(1-c/N)^{(1-t)N-1} \simeq \exp\{-(1-t)c\}$, and a non-isolated one with probability $1-\exp\{-(1-t)c\}$. Only the latter vertices are incorporated into the first-descent VC because, within the algorithm, in this case a vertex is first *uncovered*. Consequently, the expected number of available covering marks becomes

$$\begin{aligned} X(t) &= X - N\int_0^t dt'(1-\exp\{-(1-t')c\}) \\ &= X - Nt + N\frac{e^{-(1-t)c} - e^{-c}}{c}\,. \end{aligned} \tag{8.1}$$

The first descent thus describes a trajectory in the c, x plane,

$$c(t) = (1-t)c \tag{8.2}$$

$$x(t) = \frac{X(t)}{N(t)} = \frac{x-t}{1-t} + \frac{e^{-(1-t)c} - e^{-c}}{(1-t)c}\,. \tag{8.3}$$

The results are presented in Fig. 8.3. There are two crucially distinct cases: for a large enough initial value of x, $x(t)$ reaches the value one at a certain rescaled time $t < 1$. At this point, all remaining *free* vertices may be covered with the available covering marks, thus the graph is consequently proven to be coverable after having visited tN nodes of the configuration tree. With decreasing (initial) size x of the cover, the algorithm hits $x(t') = 1$ at later and later times t'. The limiting case can be obtained from Eq. (8.3) by demanding $x(t') = 1$, multiplying by $(1-t')$, setting $t' = 1$ and resolving with respect to $x = x(0)$. One obtains that hitting $x(t') = 1$ happens, as long as the starting point (x, c) is situated above the line

$$x_b(c) = 1 + \frac{e^{-c} - 1}{c}\,. \tag{8.4}$$

Below $x_b(c)$, $x(t)$ already vanishes at a positive value $c(t) > 0$. The allowed vertex-cover size thus becomes exhausted during the first descent before all edges are covered. So the branch-and-bound algorithm has to backtrack, and, intuitively, exponential solution times have to be expected.

With this observation, we have located the dynamical transition between phases A and B. For $x > x_b(c)$ the solution time is almost surely linear, exponential times appear typically only below $x_b(c)$. The specific location of this transition line depends strongly on the heuristic used in the algorithm. If we use a better heuristic, e. g., if we preferentially select vertices of high vertex degree and cover these, the transition line will move towards lower values of x,

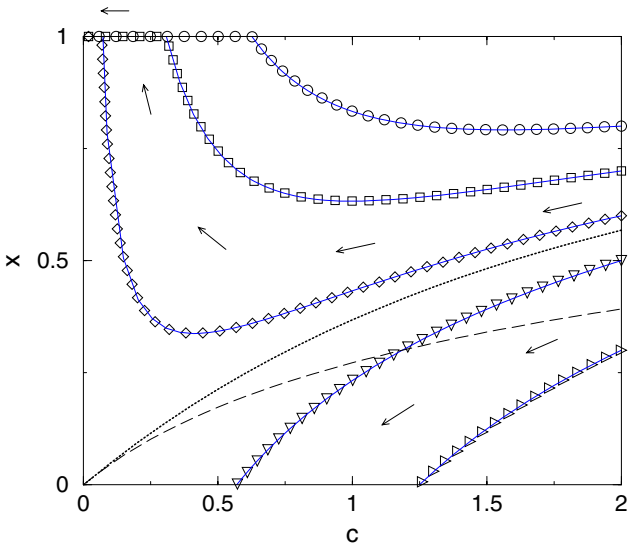


Figure 8.3: Trajectories of the first descent in the (c, x) plane. The full lines represent the analytical curves, and the symbols numerical results of one random graph with 10^6 vertices, $c(0) = c = 2.0$ and $x(0) = x = 0.8$, 0.7, 0.6, 0.5 and 0.3. The trajectories follow the sense of the arrows. The dotted line $x_b(c)$ separates the regions where this simple algorithm finds a cover from the region where the method fails. No trajectory crosses this line. The long dashed line represents the true phase boundary $x_c(c)$, as obtained in Chap. 7. Instances below that line are not coverable.

thus increasing the region of linear computational complexity. The analytical description of such algorithms becomes, however, much harder than the one presented here. The dynamics leads out of the ensemble of random graphs. The decimated graphs are no longer describable by their order and average degree alone; one has to keep track of more complicated quantities in order still to characterize the graph evolution. This case will be discussed in the second half of the chapter using a specific example.

8.1.4 The backtracking time

The dynamics in the backtracking regime is highly complicated, the algorithm has to revise decisions which were made a long, i. e. exponential, time ago. The dynamics becomes thus highly non-Markovian. At first sight, it seems hopeless to analyze this dynamics directly. A breakthrough was, however, obtained recently [4] for the satisfiability problem. The crucial idea is to consider the full configuration tree generated level by level, without taking into account at which node in the tree the algorithm passes and at which moment. Hence one is interested only in the *number* of nodes in each level. This is a feasible approach, because the

total computational time is given by the number of nodes, but it is not influenced by the actual order in which they are traversed.

We will now see that it is the knowledge of the phase diagram, as obtained in Chap. 7, that allows us to calculate the running time analytically. This again shows how knowledge obtained for a physical system can be useful in order to learn something about a computer science object, i. e., an algorithm. In this case, also in order to calculate the solution time also for $x < x_b(c)$, we combine equations (7.41) and (8.2). We have also included $x_c(c)$ into Fig. 8.3. For $x < x_b(c)$, the trajectory of the first descent crosses the phase transition line at a certain rescaled time $\tilde{t}$ at $(\tilde{c}, \tilde{x})$. There the generated random subgraph of $\tilde{N} = (1-\tilde{t})N$ vertices and average connectivity $\tilde{c}$ becomes uncoverable by the remaining $\tilde{x}\tilde{N}$ covering marks, i. e., the crossing point of the trajectory and the static transition line $x_c(c)$ is exactly the point marked by the arrow in Fig. 8.2.B. Note that $\tilde{t} = 0$ for $x < x_c(c)$, i. e., if we already start with an uncoverable graph. To prove this uncoverability the algorithm has to completely backtrack the subtree. This part of the algorithm obviously contributes the exponentially dominating part to the solution time. In the following we may thus concentrate completely on the generated subgraph, skipping "sub-" in terms like subgraph, subtree, subproblem etc.

Numerical simulations show that the exponential solution times approach a log-normal distribution of large N. Hence, the typical solution time $e^{N\tau(x|c)}$ follows from the quenched average $\tau(x|c) = \lim_{N\to\infty} \overline{\ln(t_{bt}(\tilde{G},\tilde{x}))}/N$ where $t_{bt}(\tilde{G},\tilde{x})$ is the backtracking time for the generated uncoverable instance $\tilde{G} \in \mathcal{G}(\tilde{N}, \tilde{c}/\tilde{N})$, and the over-bar denotes the average over the random graph ensemble. Solution times, as already mentioned above, are measured as the number of nodes visited by an algorithm. Since the leaves are also visited nodes, t_{bt} exceeds the number $\mathcal{N}_l$ of leaves. As the depth of the configuration tree is at most $\tilde{N}$, we also have $t_{bt} \leq \tilde{N}\mathcal{N}_l$. This means that we here just consider the last layer of the configuration tree. Since the $\tilde{N} \in \mathcal{O}(N)$ contribution is negligible compared with the exponentially growing number of leaves, the exponential time contribution is thus given by

$$\tau(x|c) \leq \lim_{N\to\infty} \frac{1}{N} \overline{\ln(\mathcal{N}_l(\tilde{G},\tilde{x}))} \,. \tag{8.5}$$

We have consequently reduced the problem of calculating the backtracking time to an entropic calculation which can be achieved using the tools of equilibrium statistical mechanics. The number of leaves is trivially bounded from above by $\binom{\tilde{N}}{\tilde{x}\tilde{N}}$, i. e., by the number of possible placements of the $\tilde{x}\tilde{N}$ covering marks on the $\tilde{N}$ vertices. Using Stirling's formula Eq. (5.15), and with $\tilde{N} = (1-\tilde{t})N = \frac{\tilde{c}}{c}N$, we thus find

$$\tau(x|c) \leq -\frac{\tilde{c}}{c}\left(\tilde{x}\ln\tilde{x} + (1-\tilde{x})\ln(1-\tilde{x})\right) \,. \tag{8.6}$$

This time is realized by our algorithm if no bound is used, i. e., if all branches of the configuration tree are visited until the covering marks are exhausted. Instead, when using the simple bound, our algorithm does not mark any two neighboring vertices simultaneously as uncovered. This excludes most of all the $\binom{\tilde{N}}{\tilde{x}\tilde{N}}$ above-mentioned configurations, leaving only an exponentially small fraction. So the simple bound already causes an exponential speedup. The number of leaves fulfilling our criterion can be characterized easily. Imagine a certain

leaf is reached at level $\kappa\tilde{N}$ of the configuration subtree. Then, our algorithm has constructed a VC of the subgraph $\tilde{G}_\kappa$ consisting of the $N^\star = \kappa\tilde{N}$ visited vertices because edges between these are not allowed to stay uncovered. Due to the random order of levels in the configuration tree, this subgraph is again a random graph with still the same edge probability $p^\star = p = c/N = \tilde{c}/\tilde{N}$, hence from the graph ensemble $\mathcal{G}(N^\star, p^\star) = \mathcal{G}(\kappa\tilde{N}, \tilde{c}/\tilde{N})$. This means the average connectivity is $c^\star = N^\star p^\star = \kappa(1-t)Nc/N = \kappa\tilde{c}$. We may thus conclude that the number of leaves at level $\kappa\tilde{N}$ equals the total number $\mathcal{N}_{vc}(\tilde{G}_\kappa, \tilde{x}\tilde{N})$ of VCs of $\tilde{G}_\kappa$ using $\tilde{x}\tilde{N} = \frac{\tilde{x}}{\kappa}N^\star$ covering marks. Here the exponential speedup becomes visible in the calculation. We only have to calculate the number of possible VCs instead of the much larger number of possible distribution of covering marks. Summing over all possible values of κ leads to the saddle point

$$\tau(x|c) = \max_\kappa \left[\lim_{N\to\infty} \frac{1}{N} \overline{\ln \mathcal{N}_{vc}(\tilde{G}_\kappa, \tilde{x}\tilde{N})} \right]. \tag{8.7}$$

The average of $\ln \mathcal{N}_{vc}$ over the random graph ensemble can be calculated using the replica trick. In order to avoid further technicalities, we use the annealed bound $\overline{\ln \mathcal{N}_{vc}} \leq \ln \overline{\mathcal{N}_{vc}} \equiv N s_{an}$, where s_{an} is the annealed entropy per vertex. This provides a very good approximation. The latter quantity was calculated in Sec. 7.2 within the first-moment method. When using Eq. (7.3) for a graph of connectivity $c^\star$ with $N^\star$ vertices, when using a relative fraction $x^\star = \tilde{x}/\kappa$ of covering marks, we thus get

$$\begin{aligned} \tau(x|c) &\simeq \max_{\kappa=\tilde{x},\ldots,1} \left[\frac{\tilde{c}}{c} \kappa s_{an}\left(\frac{\tilde{x}}{\kappa}, \tilde{c}\kappa\right) \right] \\ s_{an}(x^\star, c^\star) &= -x^\star \ln x^\star - (1-x^\star)\ln(1-x^\star) + \frac{c^\star}{2}\ln(\tilde{x}[2-x^\star]) \end{aligned} \tag{8.8}$$

where $\tilde{x}$ and $\tilde{c}$ follow from the crossing point of (8.2) with $x_{\rm c}(c)$. In Fig. 8.1 this result is compared with numerical simulations. Due to the exponential time complexity the system sizes which can be treated are of course much smaller than for the study of the first descent. In order to eliminate strong logarithmic finite-size dependencies, we have also used the number of leaves as a measure of the running time in this plot; cross-checks using the number of visited nodes in the configuration trees also show the expected behavior. Clear consistency of numerical and analytical data is found. One also finds that the computational complexity is maximal at $x = x_{\rm c}(c)$ for both algorithms with or without bound, as described by Eqs (8.6) and (8.8).

8.1.5 The dynamical phase diagram of branch-and-bound algorithms

These results imply a very intuitive picture for the different regimes of the typical-case computational complexity. It is expected to reflect the essential features of the behavior of more complicated algorithms. We can distinguish three different phases.

- *The easy coverable phase:* If x is large enough, i. e., if the global constraint on the cardinality of the VC is not very strong, even simple heuristics are able to find a solution. The configuration tree is typically given by a single descent, which terminates in a VC

of the desired size. In this phase, the algorithm is thus able to almost always solve the decision problem in a *linear* number of steps. Remember that also in this phase any random graph may be generated with non-zero probability, i. e., the *worst-case* running time of the algorithm is still expected to scale exponentially with the graph order N. These hard instances are, however, extremely rare.

- *The hard coverable phase:* If x is slightly smaller, the heuristic is no longer able to directly output a VC containing at most xN vertices. The first descent finishes in a contradiction. All covering marks are exhausted, but some of the edges remain uncovered. Still, there exist VCs of the desired size, they are just harder to find. In its first descent into the tree, the algorithm at a certain point crosses the phase transition between the coverable and the uncoverable phase, i. e., an uncoverable subproblem is generated. The algorithm has to backtrack, it performs an exponential number of steps before it realizes and corrects unfavorable choices. The solution time therefore becomes exponential. It becomes larger and larger if we further decrease x, because the starting point (x, c) moves towards the phase boundary, hence the unsolvable subproblems grow in size.

  The transition line between these two cases is strongly algorithm dependent. A better heuristic choice of vertex order and state in the first descent, shifts the onset of exponential computation times towards the coverable–uncoverable transition.

- *The hard uncoverable phase:* If we further decrease x, the system becomes uncoverable from the very beginning. This has to be shown by complete backtracking. The solution time is thus exponential, but it decreases again with decreasing x. A smaller number of covering marks is exhausted earlier, and the depth of the search tree becomes smaller. The hardest problems are thus to be found exactly at the phase boundary, where structurally very similar coverable and uncoverable instances coexist.

Related to the depicted scenario, there are mainly two different possible ways of improving this class of complete algorithms:

(i) More sophisticated heuristics for the first descent allow the onset of exponential complexity to be shifted towards $x_c(c)$. This will be the subject of the following section.

(ii) The second possibility for improving algorithms is given by the inclusion of more elaborated bounds into the configuration tree. These result in an exponential speedup of the algorithm in the hard phases, as we have already seen for the simple bound used in our algorithm.

A very efficient implementation of *(ii)* was given in Sec. 6.3. It uses a simple upper bound on the number of edges which are coverable by xN vertices. The number of these edges is at most equal to the sum of all degrees of the covered vertices, it can be smaller due to double covering some edges by both end-vertices. An upper bound on the number of coverable edges is thus given by the sum over the xN highest vertex degrees. If this number is smaller than the total edge number, the graph is obviously uncoverable with xN vertices.

Applying this bound to our initial random graph with its Poissonian degree distribution $p_d = e^{-c}c^d/d!$, the xN highest-degree vertices, having degree $d \geq d_{min}$ are described by the

equation

$$x = \alpha p_{d_{min}} + \sum_{d=d_{min}+1}^{\infty} p_d \tag{8.9}$$

which also determines the fraction $\alpha \in (0,1)$ of vertices of degree d_{min} which are to be included into the selection of the xN vertices. The number $m_{max}N$ of maximally coverable edges is thus bounded from above by

$$m_{max} \leq \alpha\, d_{min}\, p_{d_{min}} + \sum_{d=d_{min}+1}^{\infty} d\, p_d\ . \tag{8.10}$$

In a total cover, we request $m_{max} = c/2 = M/N$. If the resulting inequality is violated, we know from the very beginning that no VCs of cardinality xN exist. We do not even need to enter in a configuration tree, which thus formally consists of a single node. In this case, the problem is again easy. This allows us to add a new dynamical phase:

- *The easy uncoverable phase:* If the initial x is small enough, the bound of the algorithm may already prove the non-existence of a VC of the desired size for the initial graph. In this case the configuration tree consists of a single node, and the problem again becomes easy.

  Note that the transition between the hard and the easy uncoverable phase again depends crucially on the algorithm used, and so does the very existence of the latter phase. In the simple algorithm studied before, the hard phase extends directly to $x = 0$.

This allows us to draw schematically a phase diagram for the dynamical behavior of complete backtracking algorithms, as shown in Fig. 8.4.

8.2 The dynamics of generalized leaf-removal algorithms

This section is dedicated to the first of the two possible ways to improve algorithms mentioned above. One should use more sophisticated heuristics when choosing in which order the vertices are considered in the algorithm. From the point of view of the decision problem, they enlarge the linear-time solvable region. From the point of view of optimization, improved heuristic algorithms allow us to efficiently construct smaller vertex covers. Given that constructing a minimum VC is NP-hard, it is expected to require exponential time even if the heuristic part of the algorithm is highly optimized. We are thus restricted to small instances. For practical applications it may, however, be necessary to find a close-to-optimal solution in short time, and at this point good heuristics are of crucial interest.

The study of heuristic algorithms is also interesting from a more theoretical point of view. Their analysis allows us to construct *rigorous bounds* on the location of phase transitions, which complement the statistical mechanics approach from a mathematical point of view.

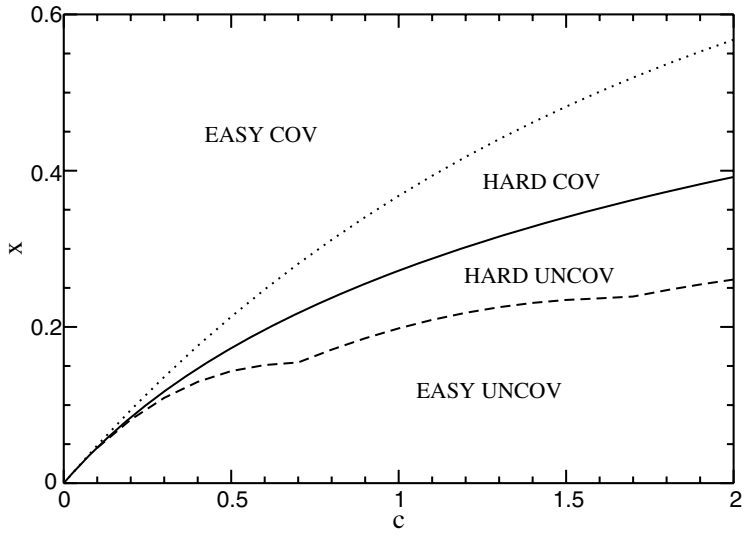


Figure 8.4: Dynamical phase diagram for branch-and-bound algorithms: The lines mark the dynamic (hard/easy) and static (cov/uncov) transitions in the behavior of branch-and-bound algorithms. The dotted line results from the simple analyzed heuristics and follows Eq. (8.4), the dashed line from the application of the bound given in Eqs (8.9) and (8.10). The non-analyticities in the latter curve result from the points where d_{min} changes. The position of both curves varies from algorithm to algorithm, depending on the heuristics and the bound used. The full line shows $x_c(c)$ as calculated in the previous chapter, it marks also the point of highest time complexity for each c. Even if the branch-and-bound algorithms have everywhere an exponential worst-case running time, the typical solution time becomes exponential only in the strip between the dashed and the dotted lines.

We have already seen in Sec. 3.1.5 that the existence of a q-core is closely related to the q-colorability of a graph, and by solving the graph reduction dynamics given by the linear-time q-core algorithm, we have found an upper bound on the chromatic number of random graphs. In a similar way, the results presented here will allow us to construct upper bounds on the size of vertex covers on random graphs, and similar methods are also used in various other optimization and decision problems.

8.2.1 The algorithm

Most heuristic algorithms exploit conjectured correlations between the local structure of the problem, i. e., in our case of the underlying graph, and the optimal solutions. The statistical mechanics analysis unveils one exploitable correlation. This is that low-degree vertices are more likely to be uncovered, while high-degree vertices are more likely to be covered, even if there is a finite fraction of vertices which are exceptions to this rule.

This correlation has already been exploited in the definition of the leaf-removal algorithm in Sec. 6.5. The argument there was the following. Take a vertex of degree one (a leaf), either

this vertex or its neighbor have to be covered to cover the connecting edge also. Covering the neighbor possibly covers more edges, thus you cannot make a mistake by covering it when you want to construct one minimum vertex cover. Both vertices and all incident edges can be removed from the graph, and the procedure is iterated. If this algorithm is successful and produces a vertex cover of the full graph, the latter is guaranteed to be a minimum one. If, however, the algorithm stops because no leaves are left before the full graph is covered, the remaining graph has to be covered by another method, e. g., by a branch-and-bound algorithm.

If one wants to find a full vertex cover by just one (heuristic) algorithm, the algorithm has therefore to be generalized. With the additional scope of remaining simple enough to be analyzable by analytical means, this is done in the following way. We have an initially uncovered graph $G = (V, E)$. In every algorithmic step a vertex i is chosen, called the *central* vertex here, and as in leaf-removal, it is uncovered, while all neighbors are covered. Then the central vertex i, its neighbors and all edges incident to these vertices are removed from G. The algorithm stops when no vertices are left. Obviously it has constructed a proper vertex cover.

The main heuristic concerns now the question, of how the central vertex i is selected. It exploits the above-mentioned correlation between vertex degree and covering state of an arbitrary vertex. The central vertex becomes uncovered. Since, in an optimal vertex cover, low-degree vertices are more likely to be uncovered than high-degree ones, it is therefore useful to preferentially select low-degree vertices. We therefore select the vertex i according to the degree-dependent weight $w_{\deg(i)}$, with w_d being a monotonously decreasing function of d.

Note that the degree of the remaining vertices may change whenever G is reduced. The algorithm always considers the *current* degree in the reduced subgraph, which equals the number of uncovered incident edges.

Some special cases of the algorithm have to be mentioned. The simplest case of $w_d \equiv 1$ was analyzed by Gazmuri [6] using a different technique. Leaf-removal is again obtained in the special case where $w_d = \delta_{d,1}$, i. e., where only leaves are selected. Bauer and Golinelli have shown [7], that this algorithm is able to cover almost all edges for $c < e$, thus producing a minimum vertex cover. It stops for higher connectivities if no vertices of degree one are left, leaving an extensive number of edges uncovered. This means a core percolation transition appears at $c = e$. A promising generalization is thus the following. If we choose $w_1 \gg w_d > 0$ for all $d > 1$, this algorithm will work nearly as well as the leaf-removal procedure for small connectivities, but it will also produce vertex covers for large c. The extreme case of always choosing a vertex of smallest current degree will work best on random graphs, but it is not analyzable by the techniques presented here.

8.2.2 Rate equations for the degree distribution

Graph reduction dynamics

We assume that the input to the algorithm is a random graph $G = (V, E)$ with N vertices and degree distribution p_d, and we first concentrate on the graph reduction process. The size of the constructed vertex cover will be calculated later.

In every algorithmic step, a vertex is selected with weight w_d depending only on its current degree d. Then, this vertex and all its nearest neighbors are removed from the graph. All edges incident to these vertices are removed, too. Following this procedure, a smaller graph is defined, and the algorithm is iterated. This graph reduction process is Markovian, because the action of each algorithmic steps depends only on the properties of the current reduced graph, more precisely, on the current vertex degrees and neighborhood relations.

As before, we measure the number T of the algorithmic steps in terms of the rescaled "time" $t = T/N$, starting thus at time $t = 0$, and every iteration step is counted as $\Delta t = 1/N$. Following again Wormald's method, see Sec. 3.3.5, the dynamics will be described by rate equations for the vertex-degree distribution $p_d(t)$, or equivalently for the number of vertices $N_d(t) = p_d(t)N(t)$ of degree d, where $N(t)$ denotes the total remaining number of vertices at time t. Their dynamics can be decomposed into the following elementary processes (where $\overline{(\cdot)} = \sum_{d=0}^{\infty}(\cdot)p_d(t)$ denotes the average over the current degree distribution $p_d(t)$).

- *Removal of the central vertex*: A central vertex of degree d is selected with weight w_d, i. e., with probability $w_d p_d(t)/\overline{w_d}$. The number $N_d(t)$ is thus reduced by one with this probability.
- *Removal of the neighbors*: According to the last item, the central vertex has on average $\overline{dw_d}/\overline{w_d}$ neighbors. As the degrees of neighboring sites are uncorrelated in random graphs, each of these has degree d with independent probabilities $q_d(t) = dp_d(t)/c(t)$, with $c(t) = \overline{d}$ being the current average degree, cf. Eq. (3.13).
- *Degree update of the second neighbors of the central vertex*: The edges connecting first and second neighbors are removed from the graph, too. Each neighbor of the central vertex has one average $\sum_d (d-1)q_d(t) = \overline{d(d-1)}/c(t)$ neighbors not being i. The degree of every second neighbor is thus reduced by one. In the same spirit as in case II in Sec. 3.3.5, N_d is decreased, if an edge attached to a vertex of degree d is removed, while N_d is increased if an edge attached to a vertex of degree $d+1$ is removed.

Similar to Eq. (3.20), the terms describing the change of $N_d(t)$ are products of how often the corresponding process happens, times the probability that the corresponding process affects a vertex of degree d. Hence, on average 1 central vertex, $\frac{\overline{dw_d}}{\overline{w_d}}$ neighbors and $\frac{\overline{dw_d}}{\overline{w_d}} \cdot \frac{\overline{d(d-1)}}{c(t)}$ second neighbors are affected.[1] These processes are combined to evolution equations for the *expected numbers* $N_d(t)$ of vertices with degree d at time t,

$$\begin{aligned} N_d(t+\Delta t) &= N_d(t) - \frac{w_d p_d(t)}{\overline{w_d}} - \frac{\overline{dw_d}}{\overline{w_d}} \cdot \frac{dp_d(t)}{c(t)} \\ &\quad + \frac{\overline{dw_d}}{\overline{w_d}} \cdot \frac{\overline{d(d-1)}}{c(t)} \cdot \frac{(d+1)p_{d+1}(t) - dp_d(t)}{c(t)} \,. \end{aligned} \tag{8.11}$$

The first line describes the deletion of vertices, the second the update of the degrees of the second neighbors. These equations are valid for the average trajectory, which is, however,

[1] Note that second-order effects are neglected, i. e., the case when two ore more neighbors of a second neighbor are removed.

followed with probability approaching 1 for $N = N(t = 0) \to \infty$. Macroscopic deviations appear only with exponentially small probability and are thus important for small N only.

Using Eq. (8.11), we can also calculate the evolution of the total numbers of remaining vertices, $N(t) = \sum_d N_d(t)$:

$$N(t + \Delta t) \quad = \quad N(t) - 1 - \frac{\overline{dw_d}}{\overline{w_d}} \tag{8.12}$$

In the limit of large graphs, $N \gg 1$, we change to intensive quantities by writing $N(t) = n(t)N$ and thus $N_d(t) = p_d(t)n(t)N$. Since for any extensive quantity $R(t) = r(t)N$ we have $R(t+\Delta t) - R(t) = \frac{R(t+1/N)/N - R(t/N)/N}{1/N} \to \dot{r}(t)$, i. e., differences become derivatives for the intensive quantities in the thermodynamic limit, we find consequently that

$$\begin{aligned} \dot{n}(t) &= -1 - \frac{\overline{dw_d}}{\overline{w_d}} \\ \dot{n}(t)p_d(t) + n(t)\dot{p}_d(t) &= -\frac{w_d p_d(t)}{\overline{w_d}} - \frac{\overline{dw_d}}{\overline{w_d}} \cdot \frac{dp_d(t)}{c(t)} \\ &\quad + \frac{\overline{dw_d}}{\overline{w_d}} \cdot \frac{\overline{d(d-1)}}{c(t)} \cdot \frac{(d+1)p_{d+1}(t) - dp_d(t)}{c(t)} . \end{aligned} \tag{8.13}$$

The graph reduction process is thus described by an infinite set of non-linear differential equations, where the non-linearity enters only through the time-dependent averages $\overline{(\cdot)}$. As we started with an ordinary random graph, these equations have to be solved under the initial condition

$$\begin{aligned} n(0) &= 1 \\ p_d(0) &= e^{-c}\frac{c^d}{d!} \end{aligned} \tag{8.14}$$

where $c = c(t = 0)$ equals the initial average vertex degree.

The cardinality of constructed vertex covers

Before trying to solve Eqs (8.13), (8.14) for specific choices of w_d, we give a general expression for the number $X(t)$ of vertices which are covered by the algorithm.

The selected central vertex is always uncovered, whereas its neighbors are covered. Denoting the expected number of covered vertices at time t with $X(t) = x(t)N$, we thus find

$$X(t + \Delta t) = X(t) + \frac{\overline{dw_d}}{\overline{w_d}} \tag{8.15}$$

cf. Eq. (8.12). Going again to the limit of large graphs, $N \to \infty$, this can be written as a differential equation for $x(t)$:

$$\dot{x}(t) = \frac{\overline{dw_d}}{\overline{w_d}} \tag{8.16}$$

with the initial condition $x(t = 0) = 0$. Note that this implies $\dot{n}(t) = -1 - \dot{x}(t)$, and consequently $n(t) = 1 - t - x(t)$, i. e., knowing the solution of Eqs (8.13), we immediately also know $x(t)$. We can deduce the covering time t_f either from $n(t_f) = 0$, when all vertices are removed and thus no uncovered edges can remain, or from $c(t_f) = 0$, when all edges of the graph are removed, i. e., covered, and only isolated vertices may survive. Then the number $x(t_f) = 1 - t_f - n(t_f)$ describes the relative cardinality of the constructed vertex cover. As the calculated average trajectory is almost surely followed for $N \to \infty$, this $x(t_f)$ gives an upper bound for the true minimum vertex cover size of the random graph under consideration. Note that for some choices of the selection weight w_d the algorithm may stop before one of these two criteria is reached, and an uncovered subgraph remains. This is, e. g., the case for the original leaf removal discussed below in Sec. 8.2.4.

8.2.3 Gazmuri's algorithm

The simplest solvable case is given by Gazmuri's algorithm [6]: In every step a vertex is selected completely randomly, irrespectively of its vertex degree. In our formalism, this corresponds to $w_d \equiv 1$ for all d. Equations (8.13) become simplified to

$$\begin{aligned} \dot{n}(t) &= -c(t) - 1 \qquad (8.17)\\ \dot{n}(t)p_d(t) + n(t)\dot{p}_d(t) &= -(d+1)p_d(t) + \overline{d(d-1)} \cdot \frac{(d+1)p_{d+1}(t) - dp_d(t)}{c(t)} \,. \end{aligned}$$

In the case of completely random vertex selection, the graph remains an Erdős–Rényi random graph with a Poissonian degree distribution of time-dependent mean $c(t)$,

$$p_d(t) = e^{-c(t)} \frac{c(t)^d}{d!} \qquad (8.18)$$

as can be verified easily by plugging this as an ansatz into Eq. (8.17). The left-hand side becomes

$$\text{l.h.s.} = -(c(t)+1)e^{-c(t)} \frac{c(t)^d}{d!} + n(t)\dot{c}(t) \left[-e^{-c(t)} \frac{c(t)^d}{d!} + e^{-c(t)} \frac{c(t)^{d-1}}{(d-1)!} \right] , \qquad (8.19)$$

whereas the right-hand side reads

$$\text{r.h.s.} = -(d+1)e^{-c(t)} \frac{c(t)^d}{d!} + c(t)^2 \left[e^{-c(t)} \frac{c(t)^d}{d!} - e^{-c(t)} \frac{c(t)^{d-1}}{(d-1)!} \right] . \qquad (8.20)$$

Equality for all d obviously implies equality for $d = 1$, which yields

$$n(t)\dot{c}(t) = -c(t) - c(t)^2 \qquad (8.21)$$

and thus, according to the first of Eqs (8.17),

$$n(t)\, \dot{c}(t) = c(t)\, \dot{n}(t) \,. \qquad (8.22)$$

This implies that $c(t) \propto n(t)$, and using the initial condition we find

$$c(t) = c\, n(t)\ . \tag{8.23}$$

We can thus eliminate $c(t)$ from the first of Eqs (8.17), which now becomes an almost trivial linear differential equation solved by

$$n(t) = \frac{(1+c)e^{-ct} - 1}{c} \tag{8.24}$$

where the initial condition $n(0) = 1$ has already been taken into account. The algorithm stops when the graph is completely removed, i. e., at time t_f given by $n(t_f) = 0$. We thus find

$$t_f = \frac{\ln(1+c)}{c}\ , \tag{8.25}$$

and the size of the constructed vertex cover is given by

$$x(t_f) = 1 - \frac{\ln(1+c)}{c}\ . \tag{8.26}$$

This expression provides, as discussed before, an *upper bound* to the minimum vertex cover size $x_{\rm c}(c)$. Comparing it with the first-moment lower bound, we see that there is a 2 in the factor of the logarithmic term. Given that the first-moment bound is conjectured to be exact in the large-c limit, this factor characterizes also the failure of Gazmuri's algorithm to construct a minimum vertex cover. The discrepancy between the two values is not astonishing given the simplicity of Gazmuri's algorithm. Up to now, there is, however, no analyzable linear-time algorithm producing an asymptotically better bound, and this problem is considered to be one of the big open questions in random-graph theory.

For small average degree c, the algorithm still does not perform very well, as can be seen in Fig. 8.5. This illustrates the need for a better heuristic, and we therefore change our selection weights in order to select preferentially low-degree vertices.

8.2.4 Generalized leaf removal

As we have already discussed, leaf removal is the extreme case $w_d = \delta_{d,1}$ of our algorithm. If it succeeds, the constructed vertex cover is a minimum one. It may, however, stop if no leaves are left, and in this situation it does not output a vertex cover. In order to construct a small vertex cover in this case also, the algorithm has to be modified to $w_d > 0$ for all $d > 0$. The case $w_d = A\delta_{d,1} + 1$, with $A \geq 0$, is interpolating between Gazmuri's algorithm and leaf removal: One is obtained for $A = 0$, the other for $A \to \infty$. We call this algorithm *generalized leaf removal* (GLR).

The best performance is, of course, obtained for $w_d = e^{-\alpha d}$, with $\alpha \to \infty$. There, a vertex of minimum degree is always selected and uncovered and all neighbors are covered. As long as the fraction $p_1(t)$ of vertices of degree one is non-zero, the algorithm is equivalent to the leaf-removal procedure. This is valid in particular also for $c < e$, where (almost) minimum VCs are constructed. The analysis of this algorithm, however, needs tools which go beyond the ones presented here.

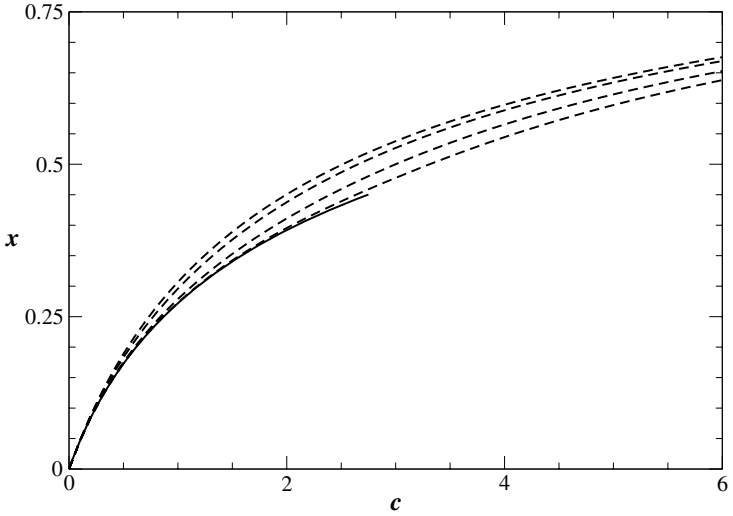


Figure 8.5: Final size $x(t_f)$ of the vertex covers produced by generalized leaf removal. The selection weight is $w_d = A\delta_{d,1} + 1$ with $A = 0, 1, 10, 100$ (dashed lines from top to bottom). The uppermost line ($A = 0$) corresponds to Gazmuri's algorithm. For a comparison, the result of the original leaf-removal algorithm is represented by the full line. For $c < e$, a minimum VC is found, whereas the algorithm fails to construct a VC for $c < e$.

Leaf removal

Let us first concentrate on the leaf-removal algorithm (LR) in its original version, i. e., on selection weights $w_d = \delta_{d,1}$. In every algorithmic step, exactly two vertices are removed from the graph G, and exactly one vertex is covered. We therefore conclude

$$\begin{aligned} n(t) &= 1 - 2t \\ x(t) &= t\,. \end{aligned} \tag{8.27}$$

The degree distribution follows, according to (8.13), from the dynamical equations

$$(1-2t)\dot{p}_d(t) = 2p_d(t) - \delta_{d,1} - \frac{\overline{d^2}}{c(t)^2} d p_d(t) + \frac{\overline{d(d-1)}}{c(t)^2}(d+1)p_{d+1}(t)\,. \tag{8.28}$$

Vertices of degree $d > 1$ are only touched if they are first or second neighbors of a vertex of degree one, in which cases they are either covered and removed, or their vertex degree is reduced by one. The degrees of neighboring vertices are statistically independent, we thus expect $p_d(t)$ to keep its Poissonian shape for all $d > 1$. Therefore we try the ansatz

$$p_d(t) = \gamma(t) e^{-\kappa(t)} \frac{\kappa(t)^d}{d!} \qquad \forall d > 1\,. \tag{8.29}$$

Since we know that $\sum_{d\geq 0} e^{-\kappa(t)}\kappa(t)^d/d! = 1$, we have $\sum_{d>1} e^{-\kappa(t)}\kappa(t)^d/d! = 1 - e^{-\kappa(t)} - e^{-\kappa(t)}\kappa(t)$. Together with the global normalization $\sum_d p_d(t) = 1$ we obtain

$$1 = p_0(t) + p_1(t) + \gamma(t)\left(1 - e^{-\kappa(t)}[1+\kappa(t)]\right) , \tag{8.30}$$

and only the three quantities $\gamma(t),\ \kappa(t)$ and $p_1(t)$ are independent, instead of the infinite sequence $p_0(t), p_1(t), \ldots$ Ansatz (8.29) can be plugged into Eq. (8.28) and leads, by analogy with the derivation for Gazmuri's algorithm in the previous subsection, to uniquely determined equations for those three quantities. For $p_1(t)$ we have trivially

$$(1-2t)\dot{p}_1(t) = 2p_1(t) - 1 - \frac{\overline{d^2}}{c(t)^2}p_1(t) + \frac{\overline{d(d-1)}}{c(t)^2}2\gamma(t)e^{-\kappa(t)}\frac{\kappa(t)^2}{2} , \tag{8.31}$$

whereas for any $d > 1$ the left-hand side of Eq. (8.28) becomes

$$\text{l.h.s.} = (1-2t)\left[\dot{\gamma}(t) + \gamma(t)\dot{\kappa}(t)\left(-1 + \frac{d}{\kappa(t)}\right)\right] e^{-\kappa(t)}\frac{\kappa(t)^d}{d!} , \tag{8.32}$$

and the right-hand side results in

$$\text{r.h.s.} = \gamma(t)\left[2 - d\frac{\overline{d^2}}{c(t)^2} + \frac{\overline{d(d-1)}}{c(t)^2}\kappa(t)\right] e^{-\kappa(t)}\frac{\kappa(t)^d}{d!} . \tag{8.33}$$

Since these expressions have to be equal for all $d > 1$, we can compare the coefficients of the different powers of d (the constant and, independently, the linear term $\sim d$ in the parentheses, each multiplied by the exterior factors), and find, using $\overline{d(d-1)} = \overline{d^2} - c(t)$

$$\begin{aligned}
(1-2t)\dot{\kappa}(t) &= -\kappa(t)\frac{\overline{d^2}}{c(t)^2} \\
(1-2t)\dot{\gamma}(t) &= \gamma(t)\left(2 - \frac{\kappa(t)}{c(t)}\right) .
\end{aligned} \tag{8.34}$$

We further have to express the last unknown quantities by $\gamma(t),\ \kappa(t)$ and $p_1(t)$:

$$\begin{aligned}
c(t) &= \sum_d d p_d(t) = p_1(t) + \gamma(t)\sum_{d\geq 2} d e^{-\kappa(t)}\frac{\kappa(t)^d}{d!} \\
&= p_1(t) + \gamma(t)\kappa(t)\left[1 - e^{-\kappa(t)}\right] \\
\overline{d(d-1)} &= \gamma(t)\kappa(t)^2 .
\end{aligned} \tag{8.35}$$

These relations can be plugged into the above differential equations. Technically it is, however, easier to work with $c(t)$ instead of $p_1(t)$. Deriving the expression for $c(t)$ in Eq. (8.35) with respect to time, we find

$$\dot{c}(t) = \dot{p}_1(t) + \dot{\gamma}(t)\kappa(t)\left[1 - e^{-\kappa(t)}\right] + \dot{\kappa}(t)\gamma(t)\left[1 - e^{-\kappa(t)} + \kappa(t)e^{-\kappa(t)}\right] . \tag{8.36}$$

The time derivatives on the right-hand side are removed using the previously derived relations, and we end up with a set of three closed equations:

$$\begin{aligned}(1-2t)\dot\kappa(t) &= -\kappa(t)\frac{\gamma(t)\kappa(t)^2+c(t)}{c(t)^2}\\ (1-2t)\dot\gamma(t) &= \gamma(t)\frac{2c(t)-\kappa(t)}{c(t)}\\ (1-2t)\dot c(t) &= 2c(t)-2\frac{\gamma(t)\kappa(t)^2+c(t)}{c(t)}\,.\end{aligned} \tag{8.37}$$

The initial conditions are $c(0)=\kappa(0)=c,\ \gamma(0)=1$. The equation for the average vertex degree $c(t)$ can be removed by observing

$$(1-2t)\frac{d}{dt}\left(\frac{\kappa(t)^2}{c(t)}\right)=(1-2t)\frac{\kappa(t)^2}{c(t)}\left(2\frac{\dot\kappa(t)}{\kappa(t)}-\frac{\dot c(t)}{c(t)}\right)=-2\frac{\kappa(t)^2}{c(t)} \tag{8.38}$$

which is solved by

$$c(t)=\frac{\kappa(t)^2}{(1-2t)c}\,. \tag{8.39}$$

The solution of the two remaining equations is given implicitly by

$$\begin{aligned}t &= 1-\frac{1}{2c}\left([\kappa(t)-W(c\,e^{\kappa(t)})]^2+2W(c\,e^{\kappa(t)})\right)\\ \gamma(t) &= \frac{W(c\,e^{\kappa(t)})}{(1-2t)c}\end{aligned} \tag{8.40}$$

as can be checked explicitly using Eqs (8.37). W denotes again the Lambert-W function which we have already seen in the last chapter, see Eq. (7.37). The graph is covered if this trajectory reaches $c(t_f)=0$, i. e., for $\kappa(t_f)=0$ due to Eq. (8.39). From the first of Eqs (8.40) we thus find the final time

$$t_f=1-\frac{W(c)^2+2W(c)}{2c}\,. \tag{8.41}$$

This result is only valid, if $p_d(t)\geq 0$ for all d and all $0<t<t_f$. Using Eq. (8.35), we find

$$\begin{aligned}p_1(t) &= c(t)-\gamma(t)\kappa(t)\left[1-e^{-\kappa(t)}\right]\\ &= \frac{\kappa(t)}{(1-2t)c}\left(\kappa(t)-W(c\,e^{\kappa(t)})[1-e^{-\kappa(t)}]\right)\\ &=: \frac{\kappa(t)}{(1-2t)c}\Phi(\kappa(t))\,.\end{aligned} \tag{8.42}$$

The factor of $\Phi(\kappa)$ is non-negative, it is thus sufficient to investigate $\Phi(\kappa)$ for $\kappa\geq 0$. We have $\Phi(0)=0$, and

$$\frac{d\Phi}{d\kappa}=1-\frac{W(c\,e^{\kappa})}{1+W(c\,e^{\kappa})}[1-e^{-\kappa}]-e^{-\kappa}W(c\,e^{\kappa})\,, \tag{8.43}$$

i. e., $\Phi'(0) = 1 - W(c)$. The monotonous function $W(c)$ becomes larger than 1 for $c > e$, i. e., in this case $p_1(t)$ would approach zero from negative values. This is a contradiction. We therefore conclude

$$\forall c < e: \qquad x_{LR}(c) = x(t_f) = t_f = 1 - \frac{W(c)^2 + 2W(c)}{2c}\,, \tag{8.44}$$

and this value coincides with the relative size of a minimum vertex cover calculated using tools from equilibrium statistical mechanics. For $c > e$ the algorithm gets stuck if $p_1(t) = 0$ is reached, no vertices of degree one are left, and a finite fraction of all edges remains uncovered.

Generalized leaf removal

In order to overcome this problem, we generalize the leaf-removal algorithm by modifying the selection weights to $w_d = 1 + A\delta_{d,1}$, as discussed above.

Also in this case, the Poissonian shape of $p_d(t)$ remains correct for all degrees $d > 1$, therefore ansatz (8.29) and normalization constraint (8.30) remain valid. Plugging everything into the dynamical equations (8.13), we directly arrive at

$$\begin{aligned}
\dot{n}(t) &= -\frac{1 + c(t) + 2Ap_1(t)}{1 + Ap_1(t)} \\
n(t)\dot{\kappa}(t) &= -\kappa(t)\frac{[\gamma(t)\kappa(t)^2 + c(t)][c(t) + Ap_1(t)]}{[1 + Ap_1(t)]c(t)^2} \\
n(t)\dot{\gamma}(t) &= \gamma(t)\frac{c(t) + 2Ap_1(t)}{1 + Ap_1(t)} - \kappa(t)\gamma(t)\frac{c(t) + Ap_1(t)}{[1 + Ap_1(t)]c(t)} \\
n(t)\dot{c}(t) &= c(t)\frac{1 + c(t) + 2Ap_1(t)}{1 + Ap_1(t)} - 2\frac{[\gamma(t)\kappa(t)^2 + c(t)][c(t) + Ap_1(t)]}{[1 + Ap_1(t)]c(t)}
\end{aligned} \tag{8.45}$$

with $p_1(t) = c(t) - \gamma(t)\kappa(t)[1 - e^{-\kappa(t)}]$. These equations determine the exact graph reduction dynamics for generalized leaf removal, and can be solved numerically. Due to the non-zero selection weights for all degrees, these equations do not suffer from the appearance of negative values for certain $p_d(t)$, and the algorithm always constructs a vertex cover. The number of covered vertices at algorithmic time t can be calculated from Eq. (8.16), which for our special choices of k and w_d, reads

$$\dot{x}(t) = \frac{c(t) + Ap_1(t)}{1 + Ap_1(t)}\,. \tag{8.46}$$

The relative size of the finally constructed vertex cover is, with probability one, given by $x(t_f)$, with t_f following from $c(t_f) = 0$. In Fig. 8.5, the results are presented for several values of A and compared with Gazmuri's as well as Bauer and Golinelli's algorithms.

8.3 Random restart algorithms

So far, considering the linear-time behavior of heuristic algorithms, we have concentrated our attention always on the typical, i. e. expected, trajectory in the space of (possibly generalized) random graphs. The reason is that the dynamics of heuristic algorithms is self-averaging, i. e., the probability that a single run of the algorithm follows the expected trajectory converges to one in the limit $N \to \infty$ of infinitely large graphs. Extensively large deviations, which lead to an extensively different size of the constructed vertex cover, are exponentially improbable and can therefore almost surely be neglected for one single run.

We have, however, seen that the heuristic algorithms are not capable of constructing a minimum vertex cover, typically one has to use exponential backtracking in order to find a minimizing solution. Instead of doing this, we can use an algorithm without backtracking, but which is re-running the (randomized) heuristics an exponential number of times. In this way, we will also see some rare trajectories leading to extensively smaller VCs than the typically expected one. To be more precise, if the probability of finding a VC of cardinality xN in a random graph $G \in \mathcal{G}(N, c/N)$ is given by a function

$$P(x|c) = \exp\{-N\omega(x_f|c)\} \tag{8.47}$$

which decreases exponentially with N, we have to restart the heuristic $\mathcal{N}(x|c)$ times with

$$\mathcal{N}(x|c) \simeq \frac{1}{P(x|c)} = \exp\{+N\omega(x|c)\} \tag{8.48}$$

to actually find at least one VC of size xN. To almost surely find a minimum vertex cover, we thus have to restart the heuristic $f(N) \cdot \mathcal{N}(x_c(c)|c)$ times, where $f(N)$ is an arbitrary function diverging in the limit $N \to \infty$. The crucial question is now the relation between $\omega(x_c(c)|c)$ and $\tau(x_c(c)|c)$ which, as introduced in Sec. 8.1, measures the logarithm of the backtracking time. We will find that the first quantity is smaller, which means that the random restart algorithm is exponentially faster than backtracking. This is, however, valid only in a probabilistic sense. Whereas backtracking is guaranteed to find a solution, the restart algorithm still has an exponentially small failure probability. The idea for this exponential boost goes back to [8], we follow mainly the analysis of [9] for VCs on random graphs, a similar analysis was also done for SAT in [10].

For simplicity, we concentrate on the naive heuristic underlying the complete algorithm of Sec. 8.1: We start with a random graph $G = (V, E) \in \mathcal{G}(N, c/N)$ and an empty vertex subset $U = \emptyset$. In every step, a vertex i is randomly selected from the set of existing vertices. If this vertex has degree zero, it is uncovered, i. e., we keep U unchanged and remove i from the vertex set, $V := V \setminus i$. If, on the other hand, the vertex has uncovered neighbors, we cover it, $U := U \cup i$, and remove vertex i and its incident edges from the graph, $V := V \setminus i$ and $E := E \setminus \{\{i, j\} \in E \mid j \in V\}$. Remember that the degree is thus always counted in the decimated current graph, not in the original random graph. The heuristic stops after N steps, when the full graph is removed, and outputs a VC U of size $|U| =: X = xN$.

In Sec. 8.1 we have already learned that this heuristic leads almost surely to a vertex cover of relative size

$$x_b(c) = 1 + \frac{e^{-c} - 1}{c} \,, \tag{8.49}$$

cf. Eq. (8.4). This means that

$$\lim_{N\to\infty} P(x_b(c)|c) = 1 \,. \tag{8.50}$$

Our aim is now to calculate $P(x|c)$ also for smaller $x < x_b(c)$. As in the analysis of the typical trajectory, this can be achieved by analyzing the possible trajectories $c(t), x(t)$ which describe the decimated problem at rescaled time $t = T/N$, where T is the number of already executed algorithmic steps. These trajectories are restricted by the following boundary conditions. The initial inputs to the algorithm are the full graph, i. e., $c(0) = c$, and an empty U, i. e., $x(0) = 0$. At time $t = 1$, there are no more uncovered edges, i. e., $c(1) = 0$, and U is a VC of cardinality xN, i. e., $x(1) = x$.

Let us thus assume that, after $T = tN$ steps, the subproblem is characterized by the quantities $c(t)$ and $x(t)$. The vertex number is $(1 - t)N$, since every step results in the removal of exactly one vertex from V. Let us further assume that the graph has still a Poissonian degree distribution,

$$p_d(t) = e^{-c(t)} \frac{c(t)^d}{d!} \,. \tag{8.51}$$

This is obviously a hardly controllable approximation for rare trajectories, but we expect it to be quite precise. The key quantity in our calculation is now the probability, that the $(T + 1)$st step changes the edges number $M(t) = (1 - t)c(t)N/2$ by some ΔM, and the cardinality of U by $\Delta X = 0/1$. According to the above heuristic rules we find

$$P_T^{T+1}(\Delta M, \Delta X) = e^{-c(t)} \left[\delta_{\Delta X,0}\delta_{\Delta M,0} + \delta_{\Delta X,1} \sum_{d=1}^{\infty} \delta_{\Delta M,-d} \frac{c(t)^d}{d!} \right] \,. \tag{8.52}$$

It is convenient to work with the Fourier transform of this function,

$$\begin{aligned} \hat{P}_T^{T+1}(\mu(t), \xi(t)) &= \sum_{\Delta M=-\infty}^{\infty} \sum_{\Delta X=-\infty}^{\infty} P_T^{T+1}(\Delta M, \Delta X) \, \exp\{-i\mu(t)\Delta M - i\xi(t)\Delta X\} \\ &= e^{-c(t)} \left[1 - e^{-i\xi(t)} \left(1 - \exp\{c(t)e^{i\mu(t)}\} \right) \right] \\ &= \exp\left\{ -c(t) + \ln\left[1 - e^{-i\xi(t)} \left(1 - \exp\{c(t)e^{i\mu(t)}\} \right) \right] \right\} \,. \end{aligned} \tag{8.53}$$

Since we will need this transformation for arbitrary times t, we have already introduced the, so far arbitrary, time-dependent notations $\mu(t)$ and $\xi(t)$. Let us now consider a number $\Delta T = N\Delta t$ of consecutive algorithmic steps, with $1 \ll \Delta T \ll N$, i. e., also $\Delta t \ll 1$. The total

transition probability can be written as the convolution of ΔT single-step probabilities,

$$P_T^{T+\Delta T}(\Delta M, \Delta X) = \sum_{\{\Delta M_\kappa, \Delta X_\kappa | \kappa=1,\ldots,\Delta T\}} \prod_{\kappa=1}^{\Delta T} P_{T+\kappa-1}^{T+\kappa}(\Delta M_\kappa, \Delta X_\kappa) \times \delta_{\Delta M, \sum_{\kappa=1}^{\Delta T} \Delta M_\kappa} \delta_{\Delta X, \sum_{\kappa=1}^{\Delta T} \Delta X_\kappa} . \quad (8.54)$$

Hence, in Fourier space, it becomes a simple product of the Fourier transformed probabilities. Neglecting subleading order terms in Δt which become irrelevant in the large-N limit, $x(t)$ and $c(t)$ can be considered as constant in this time interval,

$$\begin{aligned} P_T^{T+\Delta T}(\Delta M, \Delta X) &= \int_{-\pi}^{\pi} \frac{d\mu(t)}{2\pi} \int_{-\pi}^{\pi} \frac{d\xi(t)}{2\pi} \exp\{i\mu(t)\Delta M + i\xi(t)\Delta X\} \\ &\quad \left[\hat{P}_T^{T+1}(\mu(t), \xi(t))\right]^{\Delta T} \\ &= \int_{-\pi}^{\pi} \frac{d\mu(t)}{2\pi} \int_{-\pi}^{\pi} \frac{d\xi(t)}{2\pi} \exp\{i\mu(t)\Delta M + i\xi(t)\Delta X\} \\ &\quad \exp\left\{N\Delta t \left(-c(t) + \ln\left[1 - e^{-i\xi(t)}\left(1 - \exp\{c(t)e^{i\mu(t)}\}\right)\right]\right)\right\}. \end{aligned}$$

We can write further that

$$\begin{aligned} \Delta X &= N\, \dot{x}(t)\, \Delta t + \mathcal{O}(\Delta t^2) \\ \Delta M &= N\, \frac{d}{dt}\left[\frac{(1-t)c(t)}{2}\right] \Delta t + \mathcal{O}(\Delta t^2) \end{aligned} \quad (8.55)$$

and find

$$\begin{aligned} P_T^{T+\Delta T}(\Delta M, \Delta X) &= \int_{-\pi}^{\pi} \frac{d\mu(t)}{2\pi} \int_{-\pi}^{\pi} \frac{d\xi(t)}{2\pi} \\ &\quad \exp\left\{N\Delta t \left(i\mu(t) \frac{d}{dt}\left[\frac{(1-t)c(t)}{2}\right] + i\xi(t)\dot{x}(t)\right)\right\} \\ &\quad \exp\left\{N\Delta t \left(-c(t) + \ln\left[1 - e^{-i\xi(t)}\left(1 - \exp\{c(t)e^{i\mu(t)}\}\right)\right]\right)\right\}. \end{aligned}$$

Our final aim is to calculate $P(x|c) = P_0^N(\frac{c}{2}N, xN)$, which again can be calculated as the convolution of the probabilities of many time-slices Δt. The sum over all consecutive differences ΔM and ΔX accounts for the *sum over all paths* $(c(t), x(t))$ having the correct initial and final conditions $(c(0), x(0)) = (c, 0)$ and $(c(1), x(1)) = (0, x)$. It reads

$$\begin{aligned} P(x|c) &= \int_{-\pi}^{\pi} \prod_t \frac{d\mu(t)}{2\pi} \int_{-\pi}^{\pi} \prod_t \frac{d\xi(t)}{2\pi} \int_0^1 \prod_{t<1} dx(t) \int_0^c \prod_{t<1} dc(t) \\ &\quad \exp\left\{-N \sum_t \mathcal{L}(\dot{x}(t), \dot{c}(t), x(t), c(t), \xi(t), \mu(t))\Delta t\right\} \end{aligned}$$

where the products and sums run over $t \in \{1/N, 2/N, \ldots, 1\}$. The function $\mathcal{L}$ is given by

$$\mathcal{L}(\dot{x}, \dot{c}, x, c, \xi, \mu) = -i\mu \frac{d}{dt}\left[\frac{(1-t)c}{2}\right] - i\xi\dot{x} + c - \ln\left[1 - e^{-i\xi}\left(1 - \exp\{ce^{i\mu}\}\right)\right].$$

In the limit $N \to \infty$, this is formally written as a *path integral*,

$$P(x|c) = \int \mathcal{D}\mu(t) \int \mathcal{D}\xi(t) \int_{x(0)=0}^{x(1)=x} \mathcal{D}x(t) \int_{c(0)=c}^{c(1)=0} \mathcal{D}c(t) \exp\left\{ -N \int_0^1 dt\, \mathcal{L}(\dot{x}(t), \dot{c}(t), x(t), c(t), \xi(t), \mu(t)) \right\} .$$

This expression has the typical form of a transition rate in non-equilibrium statistical mechanics, or of a transition amplitude in quantum mechanics. In the latter case, the factor N in the exponential would be replaced by $i/\hbar$, where $\hbar$ is the Planck constant, and $\mathcal{L}$ would be the Lagrange function of the model. Exploiting this analogy further, the limit $N \to \infty$ can be seen as the analogue of the classical limit $\hbar \to 0$, and the saddle-point approximation becomes the analogue of the principle of minimal action in classical mechanics. In this limit, the action of the algorithm again becomes concentrated in the typical trajectory going from $(c(0), x(0)) = (c, 0)$ to $(c(1), x(1)) = (0, x)$. Note that this trajectory is not the typical trajectory of the algorithm discussed in the previous sections, the final condition $x(1) = x$ constrains the system to follow another trajectory. The "classical" trajectory is described by the Euler–Lagrange equations

$$\begin{aligned} \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{c}(t)} &= \frac{\partial \mathcal{L}}{\partial c(t)} \\ \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{x}(t)} &= \frac{\partial \mathcal{L}}{\partial x(t)} \\ \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\mu}(t)} &= \frac{\partial \mathcal{L}}{\partial \mu(t)} \\ \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\xi}(t)} &= \frac{\partial \mathcal{L}}{\partial \xi(t)} . \end{aligned}$$

The last two equations reduce to the usual saddle-point equations $0 = \partial\mathcal{L}/\partial\mu(t)$ and $0 = \partial\mathcal{L}/\partial\xi(t)$ since $\mathcal{L}$ does not depend on $\dot{\mu}$ and $\dot{\xi}$. The resulting equations are

$$\begin{aligned} \frac{(1-t)}{2} i\dot{\mu}(t) &= -1 - \frac{\exp\{-i\xi(t) + i\mu(t) + c(t)e^{i\mu(t)}\}}{1 - e^{-i\xi(t)}\left(1 - \exp\{c(t)e^{i\mu(t)}\}\right)} \\ i\dot{\xi}(t) &= 0 \\ \frac{d}{dt}\left[\frac{(1-t)c(t)}{2}\right] &= -\frac{c(t)\exp\{-i\xi(t) + i\mu(t) + c(t)e^{i\mu(t)}\}}{1 - e^{-i\xi(t)}\left(1 - \exp\{c(t)e^{i\mu(t)}\}\right)} \\ \dot{x}(t) &= 1 - \frac{1}{1 - e^{-i\xi(t)}\left(1 - \exp\{c(t)e^{i\mu(t)}\}\right)} . \end{aligned} \tag{8.56}$$

So, obviously, $\xi(t)$ is constant, and we set $A \equiv e^{-i\xi(t)}$. Further on, comparing the first and the third of Eqs (8.56), we find

$$\frac{(1-t)}{2} i\dot{\mu}(t) = -1 - \frac{1}{c(t)} \frac{d}{dt}\left[\frac{(1-t)c(t)}{2}\right] , \tag{8.57}$$

i. e.,

$$i\dot{\mu}(t) = -\frac{\dot{c}(t)}{c(t)} - \frac{1}{1-t} \tag{8.58}$$

which can be directly integrated over the interval $(0, t)$,

$$i\mu(t) - i\mu(0) = \ln\left(\frac{(1-t)c(0)}{c(t)}\right) . \tag{8.59}$$

We thus find

$$c(t)e^{i\mu(t)} = (1-t)c(0)e^{i\mu(0)} =: (1-t)B \tag{8.60}$$

with some constant B. The constants A and B will later on serve to fix the boundary conditions in $t = 1$. Plugging this into the last two of Eqs (8.56), these simplify to

$$\begin{aligned} \frac{d}{dt}\left[\frac{(1-t)c(t)}{2}\right] &= -\frac{(1-t)ABe^{(1-t)B}}{1-A(1-e^{(1-t)B})} \\ \dot{x}(t) &= 1 - \frac{1}{1-A(1-e^{(1-t)B})} . \end{aligned} \tag{8.61}$$

The right-hand side of both equations has become a pure time function, and the trajectory $(c(t), x(t))$ follows by direct integration of this expression [9],

$$\begin{aligned} c(t) &= \frac{c}{1-t} - \frac{2A}{(1-t)B}\int_{e^{(1-t)B}}^{e^B} dy\, \frac{\ln y}{1-A(1-y)} \\ x(t) &= t + \frac{1}{B(1-A)}\ln\left[\frac{(1-A)e^{-Bt} + Ae^{(1-t)B}}{1-A+Ae^{(1-t)B}}\right] . \end{aligned} \tag{8.62}$$

The first of these two equations results only after the variable change $e^{(1-t)B} \mapsto y$, the second can be checked easily by calculating the derivative of Eq. (8.62) with respect to t. The equations automatically fulfil the initial conditions, whereas the final conditions $c(1) = 0$ and $x(1) = x$ still have to be imposed, thereby fixing the constants A and B. Unfortunately these equations cannot be evaluated explicitly, one has therefore to rely on a numerical determination of A and B. Note that the special case of the typical trajectory discussed in Sec. 8.1 is given by $x(1) = x_b(c)$. It is reached for $A = 1$ and $B = c$, which can be seen by inserting $t = 1$, $B = c$ into the second of Eq. (8.62) and calculating the limit $A \to 1$ using the rules of de l'Hospital.

This solution can be used to determine the number $\mathcal{N}(x|c)$ of random restarts which has to be undertaken to find, almost surely, at least one VC of minimum size xN, with $x < x_b(c)$. Its normalized logarithm is given by the "action"

$$\frac{1}{N}\ln\mathcal{N}(x|c) = \omega(x|c) = \int_0^1 dt\, \mathcal{L}(\dot{x}(t), \dot{c}(t), x(t), c(t), \xi(t), \mu(t)) . \tag{8.63}$$

This quantity has to be evaluated numerically by plugging the solution (8.60,8.62) into the Lagrange function. The numerical evaluation can be achieved more efficiently by eliminating

first the time derivatives in $\mathcal{L}$ using the already simplified Euler–Lagrange equations (8.61). The results for $c = 2$ are presented in Fig. 8.6. There we plot the minimum VC size reached as a function of the number of restarts.

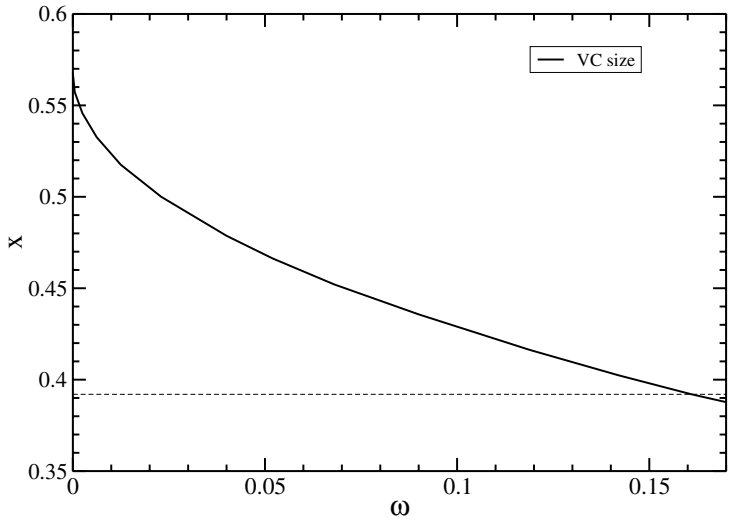


Figure 8.6: Minimum vertex cover size reached, as a function of the logarithm of the number of restarts, for random graphs of average degree $c = 2$. At $\omega = 0$, the curve starts at $x = x_b(2) \simeq 0.568$, and reaches $x_c(2) \simeq 0.392$ (represented by the dashed line) at about $\omega_c(2) \simeq 0.161$.

The curve starts at $\omega = 0$, exactly at $x_b(c)$, confirming the results for the typical dynamics of the heuristic without restarts. At some $\omega_c(c)$, the curve reaches the expected minimum VC size $x_c(c)$. For our example $c = 2$, this happens at $\omega_c(2) \simeq 0.161$. The number of restarts needed to find a minimum VC is thus, to the leading exponential order, given by $\exp\{0.161N\}$. This is to be compared with the peak complexity of the backtracking algorithm. As can be seen in Fig. 8.1, the time needed to find a minimum VC of a graph with $c = 2$ is given by about $\exp\{0.373N\}$. As an amazing consequence we find that the random-restart algorithm is exponentially faster than the backtracking algorithm – even if the algorithm is so much easier and, in particular, also much easier to implement.

Note that the curve in Fig. 8.6 also continues below the expected size of the minimum VC. At first glance, this seems to be a contradiction. How can the algorithm reach a VC which is smaller than the minimum one? It is, in fact, no contradiction at all, but a result of calculating averages over random graphs. There are exponentially rare graphs which have a macroscopically smaller minimum VC than the, almost surely, expected one. The algorithm, run on a large number of random graphs, has therefore a small probability of finding such a smaller VC, which leads to the tail of the curve in Fig. 8.6 below x_c.

At this point, we end the analysis of the dynamical behavior of known algorithms. In this chapter, we have seen the analysis of a simple complete algorithm, and also of slightly more sophisticated heuristic algorithms, as well as the possibility of exploiting large deviations by random restarts of a simple heuristic. It is, of course, possible to combine these methods in order to, e. g., calculate the typical performance of more sophisticated complete algorithms, or to characterize the large-deviation function of a less trivial heuristic. The success of this program will, however, always depend on a necessary over-simplification of the algorithms in order to keep them analytically tractable. The interest in the results obtained here comes from a slightly different motivation. These algorithms, being simple, may unveil the qualitative behavior of whole classes of similar algorithms, also including non-analyzable ones. They therefore provide a useful insight into the dynamics of algorithms, which by definition are non-equilibrium phenomena which cannot be easily understood on the basis of the statistical mechanics of equilibrium, i. e., purely in terms of the statistical properties of the solution space.

Bibliography

[1] S. Kirckpatrick, C. D. Gelatt, and M. P. Vecchi, Science **220**, 671 (1983).

[2] B. Selman, H. Kautz, and B. Cohen, in *Cliques, Coloring and Satisfiability*, ed. by D. S. Johnson and M. A. Trick, DIMACS series vol. 26 (AMS, Providence 1996).

[3] M. Weigt and A. K. Hartmann, Phys. Rev. Lett. **86**, 1658 (2001).

[4] S. Cocco and R. Monasson, Phys. Rev. Lett. **86**, 1654 (2001).

[5] S. Cocco and R. Monasson, Eur. Phys. J. B **22**, 505 (2001); Theor. Comp. Sci. A **320**, 345 (2004).

[6] P. G. Gazmuri, Networks **14**, 367 (1984)

[7] M. Bauer and O. Golinelli, Phys. Rev. Lett. **86**, 2621 (2001); Eur. Phys. J. B **24**, 339 (2001)

[8] C. P. Gomes, B. Selman, N. Crato, and H. Kautz, J. Automated Reasoning **24**, 67 (2000), C. P. Gomes, B. Selman, and H. Kautz, *Proc. AAAI-98*, 431 (Madison, WI, 1998).

[9] A. Montanari and R. Zecchina, Phys. Rev. Lett. **88**, 178701 (2002).

[10] S. Cocco and R. Monasson, Phys. Rev. E **66**, 037101 (2002).

9 Towards new, statistical-mechanics motivated algorithms

In the last two chapters, we have discussed in detail how probabilistic methods, frequently borrowed from (or being familiar in) statistical mechanics, can be applied in the analysis of typical-case properties of hard combinatorial problems. In particular, we have used these tools in order to investigate the statistical properties of minimum vertex covers on finite-connectivity random graphs, and we have characterized the typical solution behavior of simple heuristics and an even simpler complete backtracking algorithm. We have seen that these methods are useful in understanding the statistical properties of the solutions, and also in understanding and locating the onset of the exponential typical-case complexity of the algorithms.

This chapter again uses methods which were first developed within statistical physics, but with a different scope. We want to apply these methods to *single problem instances*, i. e., in the case of VC to one single sample of a random graph, and use the information gained thereby algorithmically to construct one particular optimal (or at least close to optimal) solution.

The statistical-mechanics method to be used was already introduced in Sec. 5.4.4 in the context of the ferromagnetic Ising model on a random graph. In the simplest case, we use the Bethe–Peierls iterative method, or, in more complicated cases, its generalization, the so-called cavity method. The latter was only mentioned within the discussion of the Ising model, it will be introduced here in a specific form, being suitable for the algorithm based on it.

All algorithms presented in this chapter are so-called *message-passing* algorithms (also statistical-inference algorithms). The simpler versions are equivalent to the Bethe–Peierls method, and they are well-established in various subareas of computer science and information theory, for reviews see, e. g., [1–3]. Existing also under different names, we will call this algorithm *belief propagation* (BP). It has, however, one important shortcoming: BP is only applicable if the solution space is organized in an almost trivial way corresponding to unbroken replica symmetry, as will be discussed below in this chapter. In Chap. 7 we observed, for minimum VCs on random graphs, that this is only true for a relatively small range of average degrees $c \in (0, e)$.

If the replica symmetry is broken, BP does not converge. In this case, the cavity method in its version for finite-degree models [4] can be applied. Its algorithmic single-sample version, the so-called *survey propagation* (SP), was introduced recently [5], and further developed [6, 7]. It was also applied to graph coloring [8] and finally generalized to arbitrary constraint-satisfaction problems [9].

Following the general line of this book, we develop BP and SP using the example of minimum vertex covers, particularities of k-SAT will be discussed later on in Sec. 10.4. Even if the algorithm is based on statistical-mechanics methods, we will not use specific statistical-mechanics language. Everything will be presented using a probabilistic and graph-theoretical language.

The aim is to construct a vertex cover as small as possible in polynomial time for some given graph $G = (V, E)$. The central quantities in this context will be the vertex-dependent numbers

$$\pi_i \equiv \frac{|\{U \subset V \mid U \text{ is min. VC}, i \in U\}|}{|\{U \subset V \mid U \text{ is min. VC}\}|} \tag{9.1}$$

which, for any vertex $i \in V$, equals the fraction of minimum vertex covers containing vertex $i \in V$. In probabilistic terms, it can be understood as the probability that i is covered in a randomly selected minimum vertex cover.

Once we know these quantities, we can obviously exploit them algorithmically. We know, e. g., that all vertices with $\pi_i = 1$ belong to the covered backbone, and they can be included in the VC we are aiming to construct. On the other hand all vertices $i \in V$ with $\pi_i = 0$ do belong to the uncovered backbone, and they have to be excluded from any minimum VC. The problem is slightly more involved for those vertices having π-values different from zero and one. They are contained in some vertex covers, but not in others. Since π_i gives only strictly local information, we do not know any possible quantitative restriction to the simultaneous assignment of pairs or even larger subsets of vertices. If we consider, e. g., one edge $\{i, j\} \in E$, the joint probability that both vertices are uncovered does not equal $(1-\pi_i)(1-\pi_j)$ as one might assume, naively, by considering the vertices to be independent. It obviously equals zero due to the vertex-cover constraint for the edge, i. e., one of the end-vertices has to be covered. This problem can be resolved by an iterative *decimation process*. We select a vertex of non-zero π and add it to the VC U to be constructed, and delete the vertex, as well as all its incident edges from the graph. We then recompute the π from the decimated graph, add a new vertex to U and so on, until all edges of G are covered. This reduction process is highly reminiscent of the reduction process used in the last chapter in the case of heuristic algorithms. There is, however, one major difference. In the last chapter, the selection probability of a vertex was based on a purely local information, as represented by the vertex degree, whereas the π_i do not follow from local considerations, but, as we will see in the following, from a non-trivial global self-consistency constraint instead.

9.1 The cavity graph

The following subsections will be dedicated to a method of estimating the value of π_i even for large graphs, without the need to enumerate all minimum VCs as suggested by the definition Eq. (9.1). But how can we achieve this? A simple idea could be to determine π_i from all the π_j of the neighbors $j \in N(i)$ of vertex i. The problem here is that any two of these vertices, via a path crossing i, are, at most, second neighbors of each other. So, according to the argument given above, they are strongly correlated. If, e. g., vertex i is not covered, *all* $j \in N(i)$ have

to be simultaneously covered. So the knowledge of the marginal cover probabilities π_j is obviously not sufficient to determine also the central π_i.

Remember the discussion of the Bethe–Peierls method. There we started with an exact iterative calculation of local effective fields on trees. We argued that this approach is also valid on infinitely large random graphs due to their locally tree-like structure. As a justification we considered the construction of the so-called *cavity graph*:

Definition: *Cavity graph*

Given a graph $G = (V, E)$, and a vertex $i \in V$, the *cavity graph* G_i is the subgraph of G induced by the vertex set $V_i = V \setminus i$.

In simpler words, the cavity graph is created from the full graph G by removing vertex i as well as its incident edges $\{i, j\}$ for all $j \in N(i)$. On a locally tree-like graph, or more generally on a graph with relatively long cycles, any two vertices j_a,j_b of the former neighbors of vertex i will be distant on the cavity graph G_i. The basic approximation underlying all algorithms presented in this chapter will consist in assuming statistical independence of these vertices, due to their great distance in G_i. Note that in the original graph, G, the two vertices j_a and j_b are not distant, hence statistical independence cannot be assumed, as discussed above. Only the introduction of the cavity graph allows for a simple probabilistic treatment.

Having defined the cavity graphs G_i for each vertex i, we also define the generalized probabilities

$$\pi_{j|i} = \frac{|\{U \subset V_i \mid U \text{ is min. VC of } G_i,\ j \in U\}|}{|\{U \subset V_i \mid U \text{is min. VC of } G_i\}|} \tag{9.2}$$

measuring the fraction of minimum vertex covers of the cavity graph G_i containing vertex $j \neq i$. Even if defined formally for any pair of vertices i and j, these quantities will be relevant in particular for those vertices connected by one edge in the original graph G, i. e., for $\{i, j\} \in E$.

A comment on the statistical-independence assumption has to be included at this point. We are constructing an algorithm for given realizations of graphs, i. e., finite graphs. This means that the graph's loops have finite length. The equations we are going to present in the following will therefore be only approximations to the exact values of the probabilities π_i, and the algorithm cannot guarantee really to construct a minimum vertex cover. So, even if the presented algorithm will scale only quadratically in the graph order N, it cannot be considered as an exact polynomial algorithm, and therefore does not contribute to the solution of the P-NP problem. The importance of the algorithms will be related to practical applications on large graphs, where exact methods fail due to their exponential time requirements, as already discussed in the case of simple heuristic procedures. As we will see at the end of this chapter, from numerical simulations, the message-passing procedures presented here largely outperform purely local algorithms, and therefore allow us to construct better approximations to the exact solution.

9.2 Warning propagation

The very first and simplest message-passing procedure we are going to introduce, carries the name *warning propagation* (WP). It corresponds to the Bethe–Peierls method applied exactly at zero-temperature, or, in the case of vertex cover, at infinitely high chemical potential. This limit directly restricts the vertex covers to be minimum ones.

In this case, we are not going to calculate the full marginal probabilities π_i and $\pi_{j|i}$, but only the reduced quantities

$$\tilde{\pi}_i = \begin{cases} 0 & \text{if } \pi_i = 0 \\ * & \text{if } 0 < \pi_i < 1 \\ 1 & \text{if } \pi_i = 1 \end{cases} \tag{9.3}$$

and analogously the cavity quantities $\tilde{\pi}_{j|i}$. So these quantities do not measure the exact probability for a vertex to be covered in a randomly selected minimum vertex cover. They only indicate whether it is always covered (value one), never covered (value zero) or sometimes covered and sometimes uncovered. For this last case we have introduced the unifying *joker state* $*$. Note also that this information is sufficient to be exploited algorithmically. If a vertex is assigned the joker state, it can be chosen either to be covered or to be uncovered in the current state of the graph decimation process.

In the spirit of the cavity fields introduced in Sec. 5.4.4, we construct a self-consistent set of equations for determining the $\tilde{\pi}_{j|i}$. On the basis of these, the full $\tilde{\pi}_i$ will follow.

To allow for a self-consistent construction of the $\tilde{\pi}_{j|i}$ values, different vertices have to exchange information, so-called *messages*. The next step is to introduce the simplest kind of message, the so-called warning $u_{j\to i}$ sent from a vertex j to a neighbor i. This warning incorporates the vertex cover constraint. If the vertex j is uncovered, it sends a warning $u_{j\to i} = 1$ to vertex i. This warning signifies: "Attention, to cover our connecting edge you should be covered, or I have to change state." If, on the other hand, vertex j is already covered, it sends the trivial message $u_{j\to i} = 0$ saying: "I have already covered our connecting edge, you can be either covered or uncovered." More formally, a set of messages is defined for every vertex subset $U \subset V$:

$$u_{j\to i}(U) = \begin{cases} 0 & \text{if } j \in U \\ 1 & \text{if } j \notin U \end{cases} \tag{9.4}$$

with $\{i, j\} \in E$ being an arbitrary edge. Note that each edge carries *two* messages: one sent from i to j, the other one in the opposite direction. In a proper VC, at least one of the end-vertices of each edge has to be covered, so we find that

$$U \subset V \text{ is VC of } G \quad \leftrightarrow \quad \forall \{i, j\} \in E : \; u_{i\to j}(U) \cdot u_{j\to i}(U) = 0 \,, \tag{9.5}$$

i. e., each edge has to carry at least one trivial warning. The definition of the warning can also be extended to sets $\mathcal{M}$ of vertex subsets. We define

$$u_{j\to i}(\mathcal{M}) = \min_{U\in\mathcal{M}} \; u_{j\to i}(U) \,, \tag{9.6}$$

i. e., a non-trivial message is sent if and only if vertex j is not contained in any $U \in \mathcal{M}$. This definition obviously reproduces the warning (9.4) if $\mathcal{M}$ consists of only one vertex subset. The reason for selecting the minimum in the last definition will become clear below. Using the set $\mathcal{S}_i$ of all minimum vertex covers of the cavity graph G_i as a special case, the warning $u_{j\to i}(\mathcal{S}_i)$ becomes a function of $\tilde{\pi}_{j|i}$ only. For an arbitrary but fixed edge we find

$$u_{j\to i}(\mathcal{S}_i) \equiv u_{j\to i}(\tilde{\pi}_{j|i}) = \begin{cases} 1 & \text{if } \tilde{\pi}_{j|i} = 0 \\ 0 & \text{if } \tilde{\pi}_{j|i} = * \\ 0 & \text{if } \tilde{\pi}_{j|i} = 1 \end{cases} . \tag{9.7}$$

The required minimality of the vertex cover to be constructed leads to a simple propagation of these warnings, or equivalently of the corresponding $\tilde{\pi}_{j|i}$. This can be achieved by considering how minimum vertex covers can be extended from the cavity graph to the full graph. There are three cases.

(i) There exists at least one minimum vertex cover of the cavity graph G_i where *all* $j \in N(i)$ (neighbors of i in the full graph G) are simultaneously covered. These VCs are also minimum VCs of the full graph G since all edges incident to i are already covered, so i has to be uncovered to guarantee minimality. The sizes of the minimum VCs of G_i and those of G thus coincide. In this case we find $\tilde{\pi}_i = 0$, since there are no minimum VCs of G containing i.

(ii) All minimum vertex covers of G_i leave at least two $j \in N(i)$ uncovered. Since all edges incident to vertex i have to be covered, we have to add i to the VC of G_i in order to extend it to the full graph. The VC of the full graph contains thus exactly one vertex more than those of the cavity graph, and $\tilde{\pi}_i$ equals one.

(iii) In the last, intermediate case, there is at least one minimum VC of G_i containing all but one $j \in N(i)$, but there are none containing all $j \in N(i)$. Also in this case, we have to add exactly one vertex by going from a VC of the cavity graph G_i to one of the full graph G, so the VC size grows by one. If we, however, use the VC leaving only one $j \in N(i)$ uncovered, there exists only one single uncovered edge in G. To cover it, we can select any one of its two end-vertices, i. e., either i or its neighbor. In this case, we therefore find $\tilde{\pi}_i = *$, i. e., vertex i is found to be in the joker state.

At this point, the independence assumption of all $j \in N(i)$ in the cavity graph enters into the discussion. We consider their joint probability of simultaneously being covered in a minimum VC of the cavity graph G_i, and assume this quantity to factorize into $\prod_{j\in N(i)} \pi_{j|i}$. Under this assumption, and only under this assumption, case *(i)* happens if and only if all $\pi_{j|i} > 0$, or, equivalently, if and only if all $\tilde{\pi}_{j|i} \neq 0$. Case *(ii)* happens if there are at least two vanishing $\tilde{\pi}_{j|i}$ in between the $j \in N(i)$, and the third case appears for exactly one zero $\tilde{\pi}_{j|i}$. We see that in this rule no difference between always covered and joker vertices $j \in N(i)$ exists, which justifies the use of the minimum warning in Def. (9.6). We conclude:

$$\tilde{\pi}_i = \begin{cases} 0 & \text{if } \sum_{j\in N(i)} u_{j\to i}(\tilde{\pi}_{j|i}) = 0 \\ * & \text{if } \sum_{j\in N(i)} u_{j\to i}(\tilde{\pi}_{j|i}) = 1 \\ 1 & \text{if } \sum_{j\in N(i)} u_{j\to i}(\tilde{\pi}_{j|i}) > 1 \end{cases} . \tag{9.8}$$

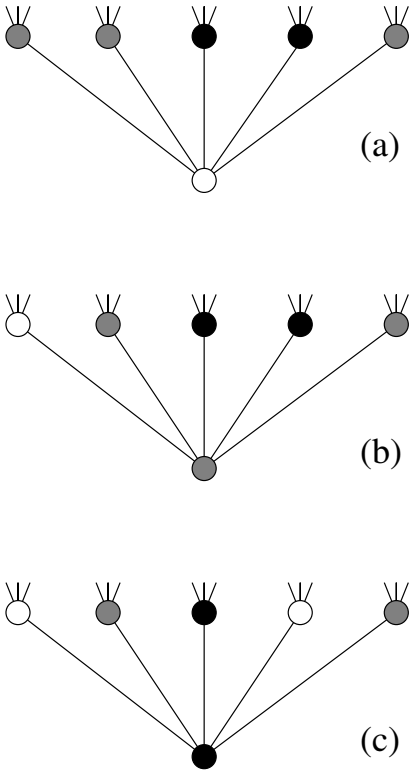


Figure 9.1: Graphical representation of Eq. (9.8), where vertex i is identified with the lower vertex in each sub-figure. The color coding of the vertices corresponds to the values of $\tilde{\pi}_i$ and the $\tilde{\pi}_{j|i}$. The value zero is represented by a white dot, the value one by a black dot, and the joker value $*$ by a gray dot. In case (a), there are no white dots between the $j \in N(i)$, so the lower vertex has not to be covered and gets the color white. If there is exactly one white dot in the upper line, the lower vertex becomes gray, cf. (b). If there are two or more white dots in the upper line, as in (c), the lower vertex is black, corresponding to an always covered vertex.

This rule is graphically represented in Fig. 9.1. Analogously to Eq. (5.63), the cavity quantities $\tilde{\pi}_{j|i}$ can now be calculated by considering the cavity graphs G_j, and by disregarding the influence of vertex i:

$$\tilde{\pi}_{j|i} = \begin{cases} 0 & \text{if } \sum_{k\in N(j)\setminus i} u_{k\to j}(\tilde{\pi}_{k|j}) = 0 \\ * & \text{if } \sum_{k\in N(j)\setminus i} u_{k\to j}(\tilde{\pi}_{k|j}) = 1 \\ 1 & \text{if } \sum_{k\in N(j)\setminus i} u_{k\to j}(\tilde{\pi}_{k|j}) > 1 \end{cases} . \tag{9.9}$$

Equations (9.7)–(9.9) are called *warning propagation* (WP). The last equation, together with Eq. (9.7), describes a system of $2|E|$ equations. Two for each edge due to the two different possible orientations of the messages. Since we have $2|E|$ variables $\tilde{\pi}_{j|i}$, the system is closed and can be solved, usually numerically, by iteration until convergence. The solution has to

plugged into Eq. (9.8) in order to calculate the values of all $\tilde{\pi}_i$. Even if this information is not yet sufficient to immediately solve the minimum VC problem, we can already read off a lot of useful information about the properties of all minimum vertex covers. The simplest quantities are the always covered and the always uncovered backbone, they are given by

$$\begin{aligned} B^{ac} &= \{i \in V \mid \tilde{\pi}_i = 1\} \\ B^{auc} &= \{i \in V \mid \tilde{\pi}_i = 0\} \,. \end{aligned} \tag{9.10}$$

A comparison with Eqs (7.42) in the chapter on the replica approach to minimum vertex covers on random graphs suggests that there is already a very close relation between the results of WP and the replica-symmetric results. In fact, in the following subsection we will rederive the latter by averaging the WP equations over the random-graph ensemble $\mathcal{G}(N, c/N)$.

Before doing so, we first show how these equations can be used to construct a vertex cover, i. e., how they can be exploited algorithmically. This is done in the following way, starting with an initial graph $G = (V, E)$ and an empty set $U = \emptyset$.

1. The $2|E|$ warnings $u_{i\to j}$ are initialized randomly.
2. Then, sequentially, edges are selected and updated first calculating the corresponding $\tilde{\pi}_{j|i}$ and $\tilde{\pi}_{j|i}$, and by replacing the old warnings with new ones according to Eq. (9.7). This update is iterated until a solution of the warning-propagation equations is found.
3. The $\tilde{\pi}_i$ are calculated from the warnings using Eq. (9.8).
4. A vertex i of maximal $\tilde{\pi}_i$ is selected (with the convention $0 < * < 1$), and it is added to U. It is removed from V, and all its incident edges are subtracted from E.
5. If uncovered edges are left, we go back to step 2, and recalculate the warnings on the decimated graph. If no edges are left, the current U is returned as a solution.

Obviously, the constructed U forms a vertex cover, since only covered edges are removed from the graph. It is also a minimum one, if the information provided by the $\tilde{\pi}_i$ was correct or, more precisely, if the warning propagation never produced a vertex of maximal $\tilde{\pi}_i$ which actually should have $\pi_i = 0$. In the presence of loops this may happen, cf., the left graph in Fig. 9.2.

Example: Success and failure of warning propagation

Consider the left side of Fig. 9.2 and, as an example, the edge from vertex 4 to vertex 3. We want to calculate $u_{4\to 3}$. Since vertex 4 has no incident edges in the cavity graph G_3, we obtain $\tilde{\pi}_{4|3} = 0$ according to Eq. (9.9), hence we have $u_{4\to 3} = 1$ according to Eq. (9.7). In the same way we get $u_{5\to 3} = 1$. In the other direction, from vertex 3 to vertex 4, the sent warning is a result of the warnings sent from the other two neighbors. The sum of incoming warnings equals at least one (in fact exactly one, when using $u_{2\to 3} = 0$), implying $\tilde{\pi}_{3|4} = *, 1$ (in fact $= *$), resulting in the "trivial", i. e., zero, outgoing warning $u_{3\to 4} = 0$. Consistency of the other warnings can be checked analogously. The final prediction is that the vertex of degree three is always covered, whereas its two lower neighbors form the

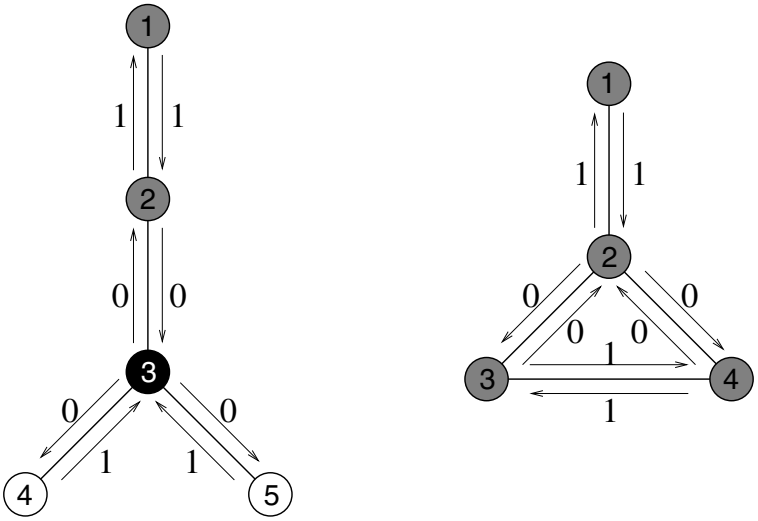


Figure 9.2: Examples for success and failure of warning propagation. In the graphs, the numbers inside the circles denote vertices, the sent warnings $u_{i\to j}(\tilde{\pi}_{j|i})$ are written close to the edges and the values of $\tilde{\pi}_i$ are represented using the color coding of Fig. 9.1.

uncovered backbone. The two upper vertices are sometimes covered, sometimes not. The exactness of this result is easily checked because the graph has only two minimum VCs which have size two.

Also, in the right side of the figure, a solution of the warning-propagation equations is represented. According to this solution, there is no backbone, which is wrong. The vertex 2 has to be covered in all minimum vertex covers. The failure of WP results from the existence of a short loop. Removing, e. g., the vertex 2, the lower vertices are still neighbors, and they are never simultaneously covered in the cavity graph, in contrast to what is assumed in warning propagation. Vertex 1 should actually be colored white, and therefore never added to a minimum vertex cover. The given solution for the warnings allows, however, to add this vertex to the VC, which is a potential source of failure of the algorithm. □

Besides the problem that the solution of the equations of warning propagation may be wrong due to the existence of short loops in the graph, there can be another problem – the iteration of the warning update may fail to converge. This can happen again due to the existence of short loops, which may lead to attractive limit cycles in the iterative warning dynamics. Another problem can appear due to the existence of *many* solutions of Eqs (9.9). We have already observed a similar situation in the case of the ferromagnet on a random graph. Below the transition temperature, two thermodynamic states exist, one with positive magnetization, and

one with negative magnetization. There we have found one solution by artificially restricting the solution space to non-negative effective fields, i. e., to the positively magnetized state. In the case of Eqs (9.9) the situation may be more drastic. The number of solutions may diverge when the graphs become larger and larger. In statistical physics we say that the replica symmetry is broken. In Chap. 7 we have already mentioned that this typically happens for random graphs of average degree $c > e$. Beyond this point, warning propagation will be of no use. We will show below how to circumvent this problem using survey propagation.

9.2.1 Back to the replica results

Before passing on to more sophisticated message-passing algorithms, we will show that warning propagation in fact reproduces the results of the replica approach to minimum vertex covers on random graphs as discussed in detail in Sec. 7.4.

In order to achieve this, we have to clarify first how the effective fields discussed in the replica approach are related to the warnings and local state variables introduced above. We define therefore

$$\begin{aligned} h_i &\equiv 1 - \sum_{j \in N(i)} u_{j \to i} \\ h_{j|i} &\equiv 1 - \sum_{k \in N(j) \setminus i} u_{k \to j} \; . \end{aligned} \tag{9.11}$$

The idea behind this definition is that a positive field $h_i = 1$ indicates that the vertex belongs to the always-uncovered backbone, see Sec. 7.4. This can only happen, if *no* neighboring vertex j signals via a message $u_{j \to i} = 1$ that it is never covered. We will now show that this definition indeed leads to the same solution as in Sec. 7.4. According to Eqs (9.8) and (9.9), these quantities allow us to determine the $\tilde{\pi}$,

$$\tilde{\pi}_i = \begin{cases} 0 & \text{if } h_i = 1 \\ * & \text{if } h_i = 0 \\ 1 & \text{if } h_i \leq -1 \end{cases} \tag{9.12}$$

and analogously

$$\tilde{\pi}_{j|i} = \begin{cases} 0 & \text{if } h_{j|i} = 1 \\ * & \text{if } h_{j|i} = 0 \\ 1 & \text{if } h_{j|i} \leq -1 \end{cases} \; . \tag{9.13}$$

This allows us to change variables in Eq. (9.7) from $\tilde{\pi}_{j|i}$ to $h_{j|i}$, we find that

$$u_{j \to i}(\tilde{\pi}_{j|i}(h_{j|i})) = \max(0, h_{j|i}) \; , \tag{9.14}$$

which, plugged into Eq. (9.11), leads to closed equations for the h-variables:

$$\begin{aligned} h_i &= 1 - \sum_{j \in N(i)} \max(0, h_{j|i}) \\ h_{j|i} &= 1 - \sum_{k \in N(j) \setminus i} \max(0, h_{k|j}) \, . \end{aligned} \tag{9.15}$$

A comparison with Eq. (7.34) suggests that the h_i are in fact the effective fields investigated within the replica approach, and the $h_{j|i}$ are cavity fields similar to those introduced within the Bethe–Peierls solution of the ferromagnet on random graphs developed in Sec. 5.4.4.

These equations are still applicable to any graph since they are defined on the actual vertices and edges. In order to reproduce the results of the replica approach, we have to constrain the graph to be randomly drawn from the Erdős–Rényi ensemble $\mathcal{G}(N, c/N)$ for large $N \gg 1$ and arbitrary, but fixed average vertex degree c. We proceed in complete analogy with the discussion in Sec. 5.4.4, and introduce the effective-field distribution

$$P(h) = \frac{1}{N} \sum_{i \in V} \delta(h - h_i) \tag{9.16}$$

as well as the cavity-field distribution

$$P_{\rm cav}(h) = \frac{1}{2|E|} \sum_{\{i,j\} \in E} \left[\delta(h - h_{i|j}) + \delta(h - h_{j|i}) \right] \tag{9.17}$$

as the global histograms over all local fields. The latter distribution is self-consistently determined by

$$\begin{aligned} P_{\rm cav}(h) &= \sum_{d=0}^{\infty} q_{d+1} \int \prod_{k=1}^{d} [dh_k P_{\rm cav}(h_k)] \; \delta \left(h - 1 + \sum_{k=1}^{d} \max(0, h_k) \right) \\ &= \sum_{d=0}^{\infty} e^{-c} \frac{c^d}{d!} \int \prod_{k=1}^{d} [dh_k P_{\rm cav}(h_k)] \; \delta \left(h - 1 + \sum_{k=1}^{d} \max(0, h_k) \right) \, . \end{aligned} \tag{9.18}$$

Here we have used again the distribution $q_{d+1} = (d+1)p_{d+1}/c$ of vertex degrees of an end-vertex of an edge randomly chosen from the set E, as defined in Eq. (3.13). In the second line, we have used the fact that this distribution, if considered for a random graph drawn from $\mathcal{G}(N, c/N)$ becomes a Poissonian of mean c. The effective field distribution follows as

$$\begin{aligned} P(h) &= \sum_{d=0}^{\infty} p_d \int \prod_{k=1}^{d} [dh_k P_{\rm cav}(h_k)] \; \delta \left(h - 1 + \sum_{k=1}^{d} \max(0, h_k) \right) \\ &= \sum_{d=0}^{\infty} e^{-c} \frac{c^d}{d!} \int \prod_{k=1}^{d} [dh_k P_{\rm cav}(h_k)] \; \delta \left(h - 1 + \sum_{k=1}^{d} \max(0, h_k) \right) \, . \end{aligned} \tag{9.19}$$

We thus find that, as in the case of the ferromagnet on random graphs, both $P(h)$ and $P_{\rm cav}(h)$ coincide due to the Poissonian nature of the degree distribution. The above self-consistency

equation determining these distributions is found to be identical to the order-parameter equation (7.34) derived before using the replica approach, which shows the equivalence between the replica-symmetric approximation and the independence assumptions forming the basis of warning propagation. Note that the derivation using the WP approach is considerably shorter than within the replica approach.

9.3 Belief propagation

After having established warning propagation as the single-sample variant of the replica-symmetric calculations, we turn back to the algorithmic point of view. The aim of the following subsections is to overcome some shortcomings of warning propagation. So far, we have restricted our attention from the initially defined probabilities π_i to the three-valued variables $\tilde{\pi}_i$ which are only able to determine whether π_i equals exactly zero, one, or lies in between. *Belief propagation* (BP) consists of a refinement of warning propagation, which takes into account the full π_i-interval $[0 \ldots 1]$.

The only equation we have to reinvestigate is the one giving a joker as the output value, i. e., case (b) in Fig. 9.1. Exactly one of the input values of the $\pi_{j|i}$ equals zero, all others are larger. Let us call the corresponding vertex j_0. The vanishing value of $\pi_{j_0|i}$ implies that $\pi_{k|j} > 0$ for all $k \in N(j_0) \setminus i$, see Fig. 9.3.

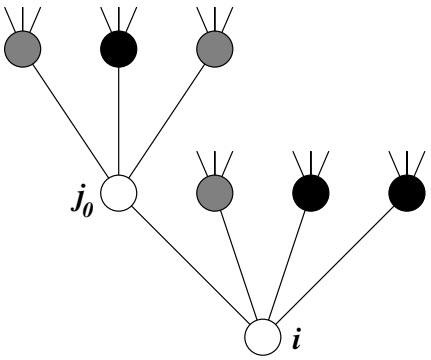


Figure 9.3: Graphical representation of the situation contributing to a non-trivial value of π_i. The vertex i has to have exactly one neighbor with $\pi_{j|i} = 0$, which here is denoted by j_0. All other $j \in N(i) \setminus j_0$ must have positive $\pi_{j|i} > 0$. Further on, $\pi_{j_0|i} = 0$ implies that all "upper" vertices $k \in N(j_0) \setminus i$ have positive $\pi_{k|j_0} > 0$, too.

Either i or j_0 have to be covered. To correctly consider both cases together with their combinatorial factors, we introduce a generalized cavity graph G_{i,j_0} by removing both i and j_0 from the graph. The question is now how far minimum vertex covers of the reduced graph can be extended to a minimum vertex cover of the full graph G. We know that only one of i and j_0 has to be covered. So, if we select i, vertex j_0 remains uncovered and we can only

take those minimum VCs of G_{i,j_0} which have all $k \in N(j_0) \setminus i$ covered. According to the above discussion these exist (none of these vertices k is a "white" one). By assuming the usual independence of these k on the cavity graph, they are simultaneously covered only in a fraction $\prod_{k\in N(j_0)\setminus i} \pi_{k|j_0}$ of all minimum VCs of G_{i,j_0}. If we select, on the other hand, j_0 to be covered, also all $j \in N(i) \setminus j_0$ have to be covered. This happens with probability $\prod_{j\in N(i)\setminus j_0} \pi_{j|i}$ in between all minimum VCs of G_{i,j_0}. Note that we have assumed that the $\pi_{j|i}$ and the $\pi_{k|j_0}$ do not change by going from simple to generalized cavity graphs, but this is again a consequence of the independence assumption. Cutting j_0 out of the cavity graph G_i does not change the properties of the $j \in N(i) \setminus j_0$. As a consequence, we can write the probability π_i in the following way:

$$\pi_i = \frac{\prod_{k\in N(j_0)\setminus i} \pi_{k|j_0}}{\prod_{k\in N(j_0)\setminus i} \pi_{k|j_0} + \prod_{j\in N(i)\setminus j_0} \pi_{j|i}} \,, \tag{9.20}$$

and, in complete analogy, for the cavity quantities

$$\pi_{i|l} = \frac{\prod_{k\in N(j_0)\setminus i} \pi_{k|j_0}}{\prod_{k\in N(j_0)\setminus i} \pi_{k|j_0} + \prod_{j\in N(i)\setminus \{l,j_0\}} \pi_{j|i}} \,, \tag{9.21}$$

by disregarding one neighbor l of i (the cavity in this case is vertex l). Note that this equation is valid if and only if there exists exactly one "white" neighbor j_0 of i with $\pi_{j_0|i} = 0$.

Plugging this modification into the equations of warning propagation, we find the following set of equations:

$$\begin{aligned}
u_{j\to i} &= \begin{cases} 1 & \text{if } \pi_{j|i} = 0 \\ 0 & \text{if } \pi_{j|i} > 0 \end{cases} \\
\pi_{i|l} &= \begin{cases} 0 & \text{if } \sum_{j\in N(i)\setminus l} u_{j\to i} = 0 \\ \frac{\prod_{k\in N(j_0)\setminus i} \pi_{k|j_0}}{\prod_{k\in N(j_0)\setminus i} \pi_{k|j_0} + \prod_{j\in N(i)\setminus \{l,j_0\}} \pi_{j|i}} & \text{if } \begin{cases} \exists j_0 \neq l \,:\, u_{j_0\to i} = 1, \\ \sum_{j\in N(i)\setminus \{j_0,l\}} u_{j\to i} = 0 \end{cases} \\ 1 & \text{if } \sum_{j\in N(i)\setminus l} u_{j\to i} > 1 \end{cases} \\
\pi_{i} &= \begin{cases} 0 & \text{if } \sum_{j\in N(i)} u_{j\to i} = 0 \\ \frac{\prod_{k\in N(j_0)\setminus i} \pi_{k|j_0}}{\prod_{k\in N(j_0)\setminus i} \pi_{k|j_0} + \prod_{j\in N(i)\setminus j_0} \pi_{j|i}} & \text{if } \begin{cases} \exists j_0 \,:\, u_{j_0\to i} = 1, \\ \sum_{j\in N(i)\setminus j_0} u_{j\to i} = 0 \end{cases} \\ 1 & \text{if } \sum_{j\in N(i)} u_{j\to i} > 1 \,. \end{cases}
\end{aligned} \tag{9.22}$$

These equations are the equations of BP. They can be exploited in the same way as WP, i. e., they can be solved for any given realization of a graph, numerically by iteration, until convergence. Unfortunately, they are limited by the same restrictions concerning graph cycles as well as the existence of many solutions and subsequent non-convergence of the iterative solution scheme.

Example: Belief propagation

As an example, we consider again the graphs from Fig. 9.2. The final values after convergence are indicated in Fig. 9.4.

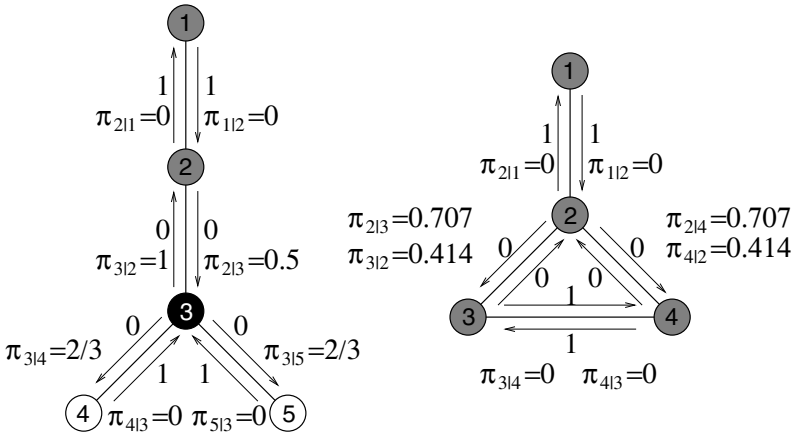


Figure 9.4: Examples for belief propagation: In the graphs, the numbers inside the circles denote vertices, the sent warnings $u_{i\to j}$ and the cavity probabilities $\pi_{i|j}$ are written close to the edges (i, j) and the values of π_i are represented using the color coding of Fig. 9.1. For the left graph one obtains $\pi_1 = \pi_2 = 0.5, \pi_3 = 1, \pi_4 = \pi_5 = 0$, while for the right graph the result is $\pi_1 = 1/(4 + 2\sqrt{2}) \approx 0.146$, $\pi_2 = (3 + 2\sqrt{2})/(4 + 2\sqrt{2}) \approx 0.854$, $\pi_3 = \pi_4 = 0.5$.

For the left graph, one can verify the values either by considering Eqs (9.22), or by direct inspection. Take, e. g., the edge $\{2, 3\}$. Without the connection to vertex 2 (i. e., in G_2), vertex 3 would be the middle vertex of a 3-vertex chain, i. e., always covered, hence with probability $\pi_{3|2} = 1$. One the other hand, without the connection to vertex 3, vertex 2 would see only his neighbor vertex 1, hence it would be covered in half of the cases, hence $\pi_{2|3} = 0.5$.

For the right graph, due to the existence of the small loop, the resulting marginal covering probabilities $\pi_{i|l}$ do not follow for the naive expectations, instead one has just to solve the Eqs (9.22). The only non-trivial resulting equations are $\pi_{2|3} = 1/(1 + \pi_{4|2})$, $\pi_{3|2} = \pi_{2|4}/(\pi_{2|4} + 1)$ for edge $\{2, 3\}$ and $\pi_{2|4} = 1/(1 + \pi_{3|2})$, $\pi_{4|2} = \pi_{2|3}/(\pi_{2|3} + 1)$ for edge $\{2, 4\}$, resulting in $\pi_{2|3} = \pi_{2|4} = 1/\sqrt{2} \approx 0.707$ and $\pi_{3|2} = \pi_{4|2} = 1/(1 + \sqrt{2}) \approx 0.414$.

Note that the cover probabilities $\pi_i > 0$ now have different values $\pi_1 = 1/(4 + 2\sqrt{2}) \approx 0.146$, $\pi_2 = (3 + 2\sqrt{2})/(4 + 2\sqrt{2}) \approx 0.854$, $\pi_3 = \pi_4 = 0.5$. Hence, if in the decimation process the vertex with the highest value of the cover probability is always covered next, here vertex 2 would be covered first, leading ultimately to a correct minimum vertex cover. □

One interesting side remark concerns the average π_i of the "gray" vertices, i. e., of non-backbone vertices. Deriving the *expected* size of a minimum VC on a (large) random graph in Chap. 7.4, we have only assumed that this average equals 1/2. Considering now the statistical symmetry between i and j_0 in Fig. 9.3, and considering that exactly one of these two vertices is covered, and the other uncovered, we immediately find the value 1/2 for the average occupation of the non-backbone vertices. Note that this is only true as an average over all non-backbone vertices. For any single vertex this can be violated. Consider, e. g., a line of four vertices $1, 2, 3, 4$, then one has for the end vertices $\pi_1 = p_4 = 1/3$, while for the inner vertices one obtains $\pi_2 = \pi_3 = 2/3$.

A last remark concerns the comparison of belief propagation to the statistical-mechanics approach of Chap. 7. We have already seen that warning propagation can be obtained easily in the limit $\mu \to \infty$ of the replica-symmetric ansatz within the latter approach. To obtain the BP equations, we can still work on the replica-symmetric level, but we have to take into account also the dominant corrections to the effective fields resulting from a finite, but large chemical potential μ, see [10] for technical details.

9.4 Survey propagation

We have already mentioned the possibility that the equations of warning or belief propagation possess a high number of solutions, and none can be found using a local iterative update scheme. The messages would try to converge to different, conflicting solutions in different regions of the graph, and global convergence cannot be achieved.

As in the case of the ferromagnet, where the two magnetized solutions correspond to two different thermodynamic states of the system, these different solutions of warning propagation also correspond to a different number of states, possibly many more than two. We now first review the physical interpretation which has emerged from the recent results about VC in the literature. Then we show one algorithm which allows us to study systems with many states, the *survey propagation* (SP) algorithm.

Speaking about minimum vertex covers, a state can be understood as a cluster of minimum vertex covers. Inside such a cluster, any two VCs are connected by at least one path via other minimum VCs, which differ stepwise only by a small number of elements (the number of these different elements stays finite in the thermodynamic limit). For two minimum VCs selected from two different clusters, no such connecting path exists, at least once an extensive step, which changes the state of $\mathcal{O}(N)$ vertices, has to be performed. Note that this distinction is, from a mathematical point of view, not well-defined for finite graphs, which are the objects of our algorithms. For finite graphs, there can also be, however, a clear separation of distance scales which allows for an identification of solution clusters see, e. g., the discussion in Sec. 6.8.

As already noted, WP and BP work well only if there is one single cluster (or a very small number of clusters). Within the replica approach, this correspond to the replica symmetric case which was discussed in detail in Chap. 7 for the average-case analysis of the vertex-cover

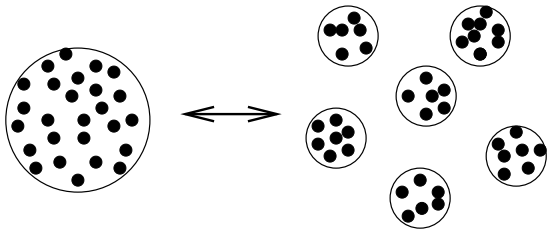


Figure 9.5: Schematic graphical representation of the organization of optimal solutions suitable for detection by WP / BP (left side) and by SP (right side). For the first case, all solutions are collected in one large unstructured cluster (or in a very small number of these clusters, as in the case of the ferromagnet), corresponding to unbroken replica symmetry. In the second case, the set of solutions is clustered into a large number of extensively separated subsets. Survey propagation corresponds to one step of replica symmetry breaking, where there is no further organization of the clusters inside the larger clusters.

problem on random graphs. There, we have already seen that the replica-symmetric solution becomes inconsistent for average vertex degrees above $c = e$, where replica symmetry breaking becomes relevant. A breaking of the replica symmetry corresponds, however, exactly to the emergence of clustering in the solution space. This effect is taken into account by the *survey propagation* (SP) algorithm, as first proposed in [5]. This algorithm is equivalent to the first step of replica symmetry breaking, where the solution clusters show no further organization. If there are clusters of clusters, etc., one has to go beyond SP.

Let us, however, assume a clustering only on one level. Instead of defining probabilities like π_i over the full solution space, we consider at the moment only one cluster at a time. Inside a cluster of minimum VCs, a vertex i may either be always covered (state 1), never covered (state 0) or sometimes covered and sometimes not (joker state $*$). Note that by using the joker state we again do not distinguish different probabilities of being covered if a vertex does not belong to the cluster backbone. This means that we treat the problem one the same level as WP, but now we allow for different solution clusters.

Hence, the assignment of this three-valued vertex state may vary from cluster to cluster. We now denote by $\hat{\pi}_i^{(1)}$ the fraction of clusters where vertex i takes state one, by $\hat{\pi}_i^{(0)}$ the fraction of clusters where vertex i takes state zero, and by $\hat{\pi}_i^{(*)}$ the fraction of clusters where vertex i takes the joker state $*$. Analogously we define the cavity quantities $\hat{\pi}_{j|i}^{(1)}$, $\hat{\pi}_{j|i}^{(0)}$ and $\hat{\pi}_{j|i}^{(*)}$ on the cavity graph G_i. A crucial assumption of SP is that clusters do not change dramatically by eliminating one vertex from the graph, i. e., by going back and forth between the full graph and the cavity graphs for different cavities.

Again, we can distinguish the three cases of how the variable states propagate inside each solution cluster. A vertex i of state 0 has to have all neighbors in states 1 or $*$ on the cavity graph G_i; a vertex i of state $*$ has to have exactly one neighbor of state 0 on the cavity graph; a vertex i of state 1 has at least two neighbors which have state 0 on the cavity graph. The statistics over all clusters can now be performed in a very simple way. The fraction of clusters

having vertex i in state 0 which by definition is $\hat{\pi}_i^{(0)}$ equals the fraction of solution clusters of the cavity graph G_i where all neighbors are in a state different from 0, and so on, for the other two states. This procedure guarantees the minimization *inside* each cluster. Note, however, that in clusters belonging to the first case no vertex has to be added to the minimal VC by stepping from the cavity graph to the full graph, whereas the VC size increases by one in the second and third case. The VCs of different clusters thus grow differently. To optimize *between* clusters, we therefore introduce a penalty e^{-y} to the last two cases. The real minimal VC should thereby be obtained in the limit $y \to \infty$, but this limit cannot be taken naively, and therefore it is useful to work at finite $y > 0$. The resulting equations are

$$\begin{aligned}
\hat{\pi}_i^{(0)} &= C_i^{-1} \prod_{j \in N(i)} (1 - \hat{\pi}_{j|i}^{(0)}) \\
\hat{\pi}_i^{(*)} &= C_i^{-1} e^{-y} \sum_{j \in N(i)} \hat{\pi}_{j|i}^{(0)} \prod_{j' \in N(i) \setminus j} (1 - \hat{\pi}_{j'|i}^{(0)}) \\
\hat{\pi}_i^{(1)} &= C_i^{-1} e^{-y} \left[1 - \prod_{j \in N(i)} (1 - \hat{\pi}_{j|i}^{(0)}) - \sum_{j \in N(i)} \hat{\pi}_{j|i}^{(0)} \prod_{j' \in N(i) \setminus j} (1 - \hat{\pi}_{j'|i}^{(0)}) \right] ,
\end{aligned} \tag{9.23}$$

and the normalization constant is given by

$$C_i = e^{-y} \left[1 - (1 - e^y) \prod_{j \in N(i)} (1 - \hat{\pi}_{j|i}^{(0)}) \right] . \tag{9.24}$$

Note that we have again made an assumption of statistical independence of the vertices j on the cavity graph. This assumption enters on two levels. First inside the cluster, when we say that j vertices of state $*$ can be covered simultaneously in a minimum VC of the cavity graph; and second in between clusters, when we factorize the joint probabilities in the upper expression.

Analogous equations are valid for the iteration of the cavity quantities, where again the influence of the cavity site has to be taken out:

$$\begin{aligned}
\hat{\pi}_{i|l}^{(0)} &= C_{i|l}^{-1} \prod_{j \in N(i) \setminus l} (1 - \hat{\pi}_{j|i}^{(0)}) \\
\hat{\pi}_{i|l}^{(*)} &= C_{i|l}^{-1} e^{-y} \sum_{j \in N(i) \setminus l} \hat{\pi}_{j|i}^{(0)} \prod_{j' \in N(i) \setminus \{j,l\}} (1 - \hat{\pi}_{j'|i}^{(0)}) \\
\hat{\pi}_{i|l}^{(1)} &= C_{i|l}^{-1} e^{-y} \left[1 - \prod_{j \in N(i) \setminus l} (1 - \hat{\pi}_{j|i}^{(0)}) - \sum_{j \in N(i) \setminus l} \hat{\pi}_{j|i}^{(0)} \prod_{j' \in N(i) \setminus \{j,l\}} (1 - \hat{\pi}_{j'|i}^{(0)}) \right] \\
C_{i|l} &= e^{-y} \left[1 - (1 - e^y) \prod_{j \in N(i) \setminus l} (1 - \hat{\pi}_{j|i}^{(0)}) \right] .
\end{aligned} \tag{9.25}$$

These are the equations for *survey propagation*. To solve them, one has to first initialize the cavity quantities arbitrarily and update them iteratively according to the second set of

equations. Once convergence is reached, the $\hat{\pi}_i^{(\cdot)}$ can be simply evaluated from the first set of equations.

Example: Survey propagation

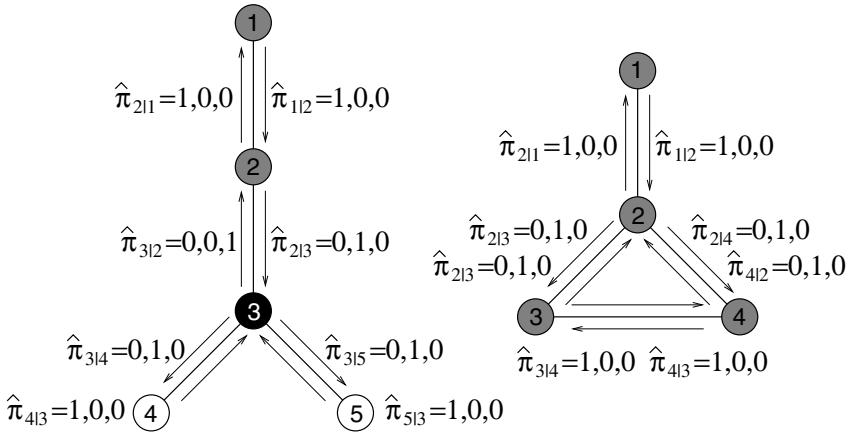


Figure 9.6: Examples for survey propagation: The cavity probabilities $\hat{\pi}_{i|j}^{(0)}, \hat{\pi}_{i|j}^{(*)}, \hat{\pi}_{i|j}^{(1)}$ are written close to the edges (i, j) and the values of π_i are represented using the color coding of Fig. 9.1. For the left graph one obtains $\hat{\pi}_1^{(*)} = \hat{\pi}_2^{(*)} = 1, \hat{\pi}_3^{(1)} = 1, \hat{\pi}_4^{(0)} = \hat{\pi}_5^{(0)} = 1$, all other values are zero, while for the right graph the result is $\hat{\pi}_i^{(*)} = 1$ and $\hat{\pi}_i^{(0)} = \hat{\pi}_i^{(1)} = 0$ for all vertices i.

As example, we consider again the graphs from Fig. 9.2. The final values after convergence are indicated in Fig. 9.6. □

Although the knowledge of all $\hat{\pi}_i^{(\cdot)}$ does not allow us to directly create a minimum vertex cover, due to the impossibility of deducing a joint probability distribution of all vertices from only the knowledge of the marginal single-vertex probabilities, nevertheless, some useful knowledge can be taken from these quantities. Using again our preferred example, we can, e. g., immediately read off the SP predictions for the two backbones of always covered and always uncovered vertices – where now always refers again to the set of all solutions, and not to single clusters. We conclude that

$$\begin{aligned} B^{ac} &= \{i \in V \mid \hat{\pi}_i^{(1)} = 1\} \\ B^{auc} &= \{i \in V \mid \hat{\pi}_i^{(0)} = 1\} \end{aligned} \tag{9.26}$$

are given by those variables which take the same state in all clusters.

To actually construct a minimum vertex cover (or an approximation due to the non-exactness of SP because of, e. g., a finite value of y, cycles in the graph or more levels of cluster organization), we have to resort again to an iterative decimation scheme. In every step, the $\hat{\pi}_i^{(\cdot)}$ are calculated, one vertex of large $\hat{\pi}_i^{(1)}$ is selected and covered. It is removed from the graph together with all incident edges, and the $\hat{\pi}_i^{(\cdot)}$ are reiterated on the decimated graph. This procedure is iterated until no uncovered edges are left, i. e., until a proper VC is constructed. Slightly different schemes of selection heuristics can be applied (select a vertex of high $\hat{\pi}_i^{(0)}$, uncover it, cover all neighbors, and decimate the graph, or take into account also the value of $\hat{\pi}_i^{(*)}$). All these heuristic rules are equally valid in the sense that, if SP is exact on a graph, they all produce one minimum VC of the same size. For real graphs, however, where the results of the SP equations are to be considered as approximations of the actual quantities defined over the set of solutions, different heuristic choices may result in VCs of different sizes. Within the numerical experiments described below, we have, however, found no preferable selection heuristic, and the fluctuations from one heuristic to another were small compared with the VC size.

9.5 Numerical experiments on random graphs

These algorithms have to be tested on real graphs. Here we show some results for random graphs of finite average connectivity even if the latter sometimes may become as large as $c = 400$. The ensemble under consideration is $\mathcal{G}(N, \frac{c}{2}N)$, i. e., the number of edges is fixed to $\frac{c}{2}N$.

In the experiments, we have first generated random graphs from this ensemble. In doing so, one has to carefully exclude multiple edges between two vertices, these are not considered in the diverse message-passing procedures. On these graphs, we have run WP, BP and SP as well as two local algorithms, one being Gazmuri's algorithm, the other, better, being generalized leaf-removal in its strongest form. In each step a vertex of minimum current degree is selected and uncovered, all neighbors are covered, and these vertices are removed from the graph. For details on these algorithms see Chaps 6 and 8 where they were discussed in great detail.

Low average degree

First we have considered graphs of low average degree $c < e$. From the two previous chapters we know that, with probability approaching one in the infinite-graph limit $N \to \infty$, the replica symmetric results are exact, and leaf removal is able to find a minimum VC. In fact, this was impressively confirmed by the numerical simulations even for graphs as small as $N = 10$. The results of SP collapsed to those of WP, i. e., the $\hat{\pi}_i$ became completely concentrated in one out of $\{0, *, 1\}$ thus reproducing a solution of cluster as given by WP. The produced vertex covers were, in most cases, of the same cardinality as those produced by generalized leaf removal. Only in a very small fraction of cases, decaying to zero for increasing graph

size, one of the algorithms produced a slightly larger VC, presumably due to rare small cycles in the underlying random graph.

High average degree

If we go to average vertex degrees $c > e$, WP and BP stop to converge, and they are no longer of any use in constructing small vertex covers. SP instead, starts to converge to a non-trivial solution which is not concentrated. This is true only for relatively small y, for large y no global convergence of SP is achieved. We have performed experiments up to $c = 400$ and $N = 6\,400$ (note that the algorithm in this case has to handle $cN = 2\,560\,000$ values of the cavity quantities $\hat{\pi}_{j|i}$). The results for fixed c and various values of N were interpolated to $N \to \infty$ in order to be comparable to asymptotically correct analytical results. For this comparison we have chosen the lower and upper bounds on the expected cardinality of minimum vertex covers as derived in the last two chapters via the first-moment method and the dynamical analysis of Gazmuri's algorithm. We have also included the Frieze formula (7.7) describing the correct asymptotic behavior for $c \gg 1$.

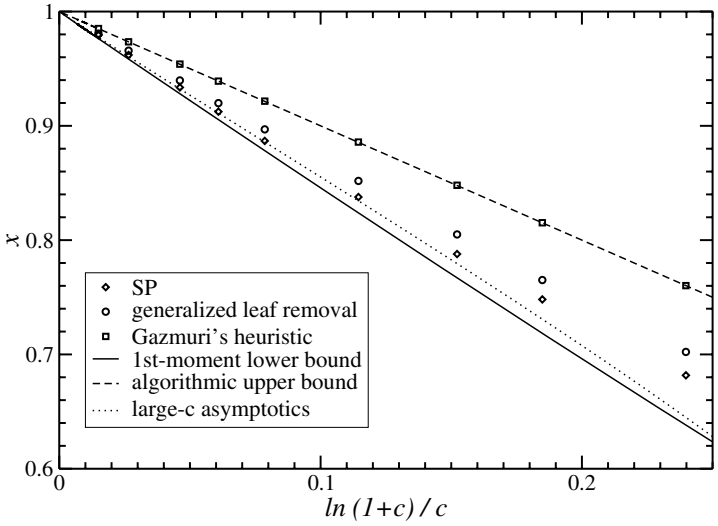


Figure 9.7: Numerical results of SP for random graphs of high, but finite average vertex degree $c \in [10, 400]$. The data are generated by running SP on graphs of fixed c, but various orders N, and by extrapolating the data to infinite graph order. The results are compared with the outcomes of various local heuristics, which are clearly outperformed by SP, and with rigorous lower and upper bounds, as well as the exact large-c asymptotics which seems to be met by SP.

The results are represented in Fig. 9.7, and they rather impressively show the performance of SP as compared to the local heuristics. Note that the exact asymptotics and the the results of SP

almost coincide for large c, which suggests that SP, in contrast to local heuristic algorithms, is able to reach VCs which, even if not guaranteed to be minimum due to the statistical-independence approximations used, will reach the correct asymptotic size.

One has, however, to mention that the non-convergence of SP for high y-values as well as further analytical results indicate that the simple clustering structure assumed in one-step replica symmetry breaking, and which lead to the SP equations, are too simple to capture correctly the real organization of the solution space structure [10, 11]. Analytical results suggest that, for $N \to \infty$, an infinite hierarchy of clusters of clusters, etc., exists, which cannot be represented by SP. On the other hand, for finite graphs, message-passing algorithms will always be of an approximate nature, such that an application is even justified if the mathematical correctness of the underlying assumptions has not completely been established. This justification results from the practical success of the algorithm, which for random graphs is shown in Fig. 9.7. Note, however, that the structure of random graphs favors the applicability of message-passing procedures which are developed for locally tree-like graphs. Applying SP to dense graphs with $c \propto N$ in fact shows that, in the presence of many short loops, SP performs only comparably to the local search heuristics. This means that, before blindly applying algorithms of the message-passing type, one should check if the graph is not dominated by dense local subgraphs. In fact, current research activity tries to overcome this limitation by, e. g., coarse-graining the underlying graph.

Bibliography

[1] B. J. Frey, *Graphical Models for Machine Learning and Digital Communication* (MIT Press, 1998).

[2] F. R. Kschischang, B. R. Frey, and H.A, Loeliger, IEEE Trans. Info. Theory **47**, 498 (2001).

[3] J. S. Yedidia, W. T. Freeman, and Y. Weiss, Technical report TR-2001-22 of the Mitsubishi Electric Research Laboratories (2002).

[4] M. Mézard and G. Parisi, Eur. Phys. J. B **20**, 217 (2001).

[5] M. Mézard, G. Parisi, and R. Zecchina, Science **297**, 812 (2002).

[6] M. Mézard and R. Zecchina, Phys. Rev. E *66*, 056126 (2002).

[7] A. Braunstein, M. Mézard, and R. Zecchina, submitted to Rand. Struct. Alg.; arXiv preprint `cs.CC/0212002`.

[8] A. Braunstein, R. Mulet, A. Pagnani, M. Weigt, and R. Zecchina, Phys. Rev. E **68**, 036702 (2003).

[9] A. Braunstein, M. Mézard, M. Weigt, and R. Zecchina, To appear in *Computational Complexity and Statistical Physics*, Editors: A. Percus, G. Istrate and C. Moore (2005).

[10] M. Weigt and A. K. Hartmann, Phys. Rev. E **63**, 056127 (2001).

[11] H. Zhou, Eur. Phys. J. B **32**, 265 (2003).

10 The satisfiability problem

In this chapter, a short summary of recent developments concerning the satisfiability problem (SAT) is given. SAT and K-SAT have already been introduced (see Chap. 4 for a formal definition). To remind you shortly, SAT is the decision problem, whether for a given Boolean formula $F(x_1, \ldots, x_n)$, there exists an assignment of the variables $x_i \in \{0, 1\}$ ("false", "true") leading to a "true" evaluation $F(x_1, \ldots, x_n) = 1$. A literal is a variable or its negation, and formulas in conjunctive normal form (CNF) consist of a conjunction of m clauses, where each clause itself is a disjunction of an arbitrary finite number of literals. The latter number equals exactly K for K-CNF formulas, as studied in K-SAT.

SAT is the most basic of all combinatorial problems. It is the first problem for which the NP-completeness has been shown, as we have explained in Chap. 4. Hence, it belongs to the most frequently studied problems in this field. It is the first NP-complete problem for which a phase transition has been observed [1, 2]. This observation has triggered the recent interest of statistical physicists in combinatorial optimization [3–5]. Finally, also the survey propagation algorithm was first developed for SAT [6, 7] on the basis of the cavity method [8] used in the statistical mechanics of diluted disordered systems.

The analytical treatment of SAT is, however, technically more involved than the treatment of vertex cover. Thus, we have used the latter problem to introduce the concepts and methods of the statistical mechanics of combinatorial optimization. Nevertheless, the satisfiability problem is of such high importance that we want to discuss the particularities of SAT. This chapter therefore gives an overview of the most basic algorithms, techniques and results. Note, however, that we recommend first reading the corresponding chapters on vertex covers, since the basic terminology as well as the main tools are introduced and elaborated there.

We first explain and analyze some fundamental algorithms to solve SAT, an exhaustive review can be found in Ref. [9]. Next, we show that random K-SAT formulas exhibit a satisfiable/unsatisfiable phase transition, which coincides with a maximum of the typical running time of an exact algorithm, and we discuss the picture arising from the statistical mechanics approach. Then we outline the approximate analysis of a stochastic local search (SLS) algorithm. At the end, we explain how warning and survey propagation have to be modified when going from VC to SAT.

10.1 SAT algorithms

As for all algorithms solving combinatorial (optimization) problems, one can distinguish between complete algorithms, which give the exact answer, or incomplete and stochastic algorithms, which give the true answer only with some probability. Stochastic algorithms are frequently faster, but it cannot be guaranteed that the correct answer is indeed obtained. In this section, we first discuss complete algorithms for the 2-SAT problem, which is polynomially solvable. Then we explain complete algorithms for the general K-SAT case, where no polynomial algorithm is known. At the end, some incomplete algorithms are presented, in particular WalkSAT. But first, we introduce a useful representation of SAT formulas, and an important auxiliary procedure: the so-called *unit-clause propagation*.

For the algorithmic treatment, we represent each CNF formula as a collection $\Gamma_0, \Gamma_1, \ldots, \Gamma_K$ of sets of clauses, where clauses of length i are contained in Γ_i. For technical reasons, the set Γ_0 contains the unsatisfied clauses.

Whenever a variable is assigned, one can evaluate the consequences on a given CNF formula: First, a clause can be removed, if the assigned value of the variable leads to a "true" literal, because the full clause is then satisfied. Second, in each clause, where the variable assignment leads to a "false" literal, the literal can be removed from the clause (i. e., a clause $c \in \Gamma_i$ has to be moved to Γ_{i-1}). This second step might generate clauses containing just one literal. Those literals can be immediately assigned values turning them "true" as well, possibly leading to further simplifications of the formula. This process of simplifying formulas is called *unit-clause propagation* (UCP). UCP will be needed for some of the following algorithms, hence we present a UCP procedure here. Note that assigning the value of a variable x is done by adding either (x) (equivalent to setting it to "true") or $(\overline{x})$ ("false") to Γ_1. The parameters are passed via *call by reference*, that means that the changes on the passed values, which are performed inside the procedure, will still be effective outside the procedure.

procedure UCP([**call by ref.**] $\Gamma_0, \Gamma_1, \ldots, \Gamma_k, U$)
begin
  while $\Gamma_1 \neq \emptyset$ **do**
  begin
    let l be a literal in Γ_1; $\Gamma_1 = \Gamma_1 \setminus \{(l)\}$;
    let x be the variable corresponding to l; $U \setminus \{x\}$;
    for $j = 1, \ldots, K$ **do**
      for all $c \in \Gamma_j$ **do**
      begin
        if $l \in c$ **then** {clause is satisfied}
          $\Gamma_j = \Gamma_j \setminus \{c\}$;
        if $\overline{l} \in c$ **then** {literal is "false"}
          $\Gamma_j = \Gamma_j \setminus \{c\}$; $\Gamma_{j-1} = \Gamma_{j-1} \cup \{c \setminus \overline{l}\}$;
      end
  end
end

An example of the operation of UCP will be given in the following subsection, where two algorithms for 2-SAT are described.

UCP can be extended, for example, by checking whether some variables occur only non-negated, or only negated in the formula. In this case we speak of a pure literal. One can immediately assign the corresponding value, and all the clauses containing the literal are satisfied and can be removed from the formula. Also, clauses that contain *all* the literals of another clause can be removed immediately. We do not go into further details here.

10.1.1 2-SAT algorithms

2-SAT can be decided in polynomial, more precisely, even in linear time [10]. We will explain a straightforward algorithm [11, 12] which was historically developed first. Then we will explain the linear-time algorithm.

The basic idea of the simple polynomial 2-SAT algorithm is to start with any one variable x, assign a value (say $x = 1$) and evaluate all logical consequences using UCP, leading to further assignments. If a contradiction occurs, i. e., a clause is evaluated to "false", then one tries the other possible assignment of x (i. e., $x = 0$) and again uses UCP. If both assignments lead to contradictions, then the formula is not satisfiable. If no contradiction occurs, one continues by assigning another yet unassigned variable. This is continued until a contradiction occurs, or until all variables have been assigned. The algorithm reads as follows. Note that assigning the value of a variable x is again done by adding either x ("true") or $\overline{x}$ ("false") to Γ_1. When calling the following algorithm, Γ_2' contains all clauses of the 2-CNF formula F and the set U contains all variables.

algorithm simple-2-SAT(Γ_2', U)
begin
  while $U \neq \emptyset$ **do**
  begin
    Let x be a variable in U;
    $\Gamma_0 = \emptyset$; $\Gamma_1 = \{(x)\}$; $\Gamma_2 = \Gamma_2'$; {try $x = 1$}
    UCP($\Gamma_0, \Gamma_1, \Gamma_2, U$);
    if $\Gamma_0 \neq \emptyset$ **then** {contradiction ?}
    begin
      $\Gamma_0 = \emptyset$; $\Gamma_1 = \{(\overline{x})\}$; $\Gamma_2 = \Gamma_2'$; {try $x = 0$}
      UCP($\Gamma_0, \Gamma_1, \Gamma_2, U$);
      if $\Gamma_0 \neq \emptyset$ **then** {contradiction !}
        print unsatisfiable; **stop**;
    end
    $\Gamma_2' = \Gamma_2$; {continue with reduced formula}
  end
  print satisfiable;
end

The reason why this works is that, since all clauses have at most two literals, *all* consequences of assigning a variable can be *immediately* evaluated, i. e., UCP runs until all consequences are considered. In UCP two possibilities can occur.

- If a literal becomes "true", then the clause can always be omitted. This holds also for clauses consisting of more than two literals, e. g., for 3-SAT.
- On the other hand, if a literal evaluates to "false", we know immediately that the other literal must become "true" to satisfy the full formula, because the clause contains only two literals (the reduced clause is moved to Γ_1 by UCP). In the general case of clauses having $j > 2$ literals, this does not result in an immediate assignment of other variables (the clause is moved to Γ_{j-1} by UCP), because several possibilities still exist of satisfying the clause.

This means that for 2-SAT, any assignment not leading to an immediate contradiction can be always kept, i. e., no backtracking beyond one level (i. e., taking back an assignment and its immediate consequences) is necessary. This works because, for 2-SAT, the satisfiability of the remaining formula does *not* depend on the value which has been chosen for x.

Example: Simple-2-SAT

As an example, we consider first the formula

$$\begin{aligned} F_1 \quad = \quad & (\overline{x}_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (x_1 \vee x_4) \wedge \\ & (x_2 \vee x_3) \wedge (\overline{x}_2 \vee \overline{x}_3)\,. \end{aligned} \tag{10.1}$$

We assume that, in the main loop of the simple-2-SAT algorithm, first $x_1 = 1$ is assigned. Hence UCP is called with $\Gamma_0 = \emptyset$ (as always), $\Gamma_1 = \{(x_1)\}$ and $\Gamma_2 = F$.

During the first iteration in UCP (see page 233), all clauses containing x_1 will drop out, i. e., the clause $(x_1 \vee x_4)$. The clauses containing $\overline{x}_1$ will be reduced in length, i. e., $(\overline{x}_1 \vee \overline{x}_2)$ and $(\overline{x}_1 \vee \overline{x}_3)$, and then be moved to Γ_1. Hence, we obtain after the first iteration in UCP:

$$\Gamma_0 = \emptyset, \quad \Gamma_1 = \{(\overline{x}_2), (\overline{x}_3)\}, \quad \Gamma_2 = \{(x_2 \vee x_3), (\overline{x}_2 \vee \overline{x}_3)\}\,.$$

Now, assume that UCP picks $\overline{x}_2$ from Γ_1. Hence the clause $(x_2 \vee x_3)$ is reduced to (x_3) and clause $(\overline{x}_2 \vee \overline{x}_3)$ is "true", and hence dropped. After the second iteration of UCP, we have:

$$\Gamma_0 = \emptyset, \quad \Gamma_1 = \{(\overline{x}_3), (x_3)\}, \quad \Gamma_3 = \emptyset\,.$$

In the third iteration, UCP might pick x_3 from Γ_1. Hence the clause $(\overline{x}_3)$ will be shortened and moved to Γ_0, indicating that a contradiction has occurred. Now Γ_1 is empty, hence UCP finishes.

In simple-2-SAT, it will be noticed that $\Gamma_0 \neq \emptyset$, hence the other choice $\Gamma_1 = \{(\overline{x}_1)\}$ is tried and again UCP is called.

During the first iteration, UCP recognizes that $(\overline{x}_1 \vee \overline{x}_2)$ and $(\overline{x}_1 \vee \overline{x}_3)$ are satisfied, hence they are removed. $(x_1 \vee x_4)$ is not satisfied, i. e., reduced. We obtain

$$\Gamma_0 = \emptyset,\ \Gamma_1 = \{(x_4)\},\ \Gamma_2 = \{(x_2 \vee x_3), (\overline{x}_2 \vee \overline{x}_3)\}\,.$$

During the second iteration, x_4 is treated. Since it is not contained in the clauses in Γ_2, no change occurs, except that x_4 is removed from Γ_1. Now Γ_1 is empty, hence UCP terminates.

Being back in simple-2-SAT, Γ_2 will be set to $\{(x_2 \vee x_3), (\overline{x}_2 \vee \overline{x}_3)\}$, and $U = \{x_2, x_3\}$. Now assume that x_2 will be chosen, i. e., $x_2 = 1$ is assigned. This means UCP is called with $\Gamma_1 = \{(x_2)\}$ and $\Gamma_2 = \{(x_2 \vee x_3), (\overline{x}_2 \vee \overline{x}_3)\}$. In the first iteration, while treating literal x_2, UCP will remove clause $(x_2 \vee x_3)$ and reduce $(\overline{x}_2 \vee \overline{x}_3)$ to $(\overline{x}_3)$. In the second iteration, UCP will assign $x_3 = 0$ and terminate.

The algorithm recognizes that all variables have been assigned values ($U = \emptyset$), hence the formula is satisfiable with $x_1 = 0$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$, being a satisfying assignment.

To see how the algorithm operates, if the formula is not satisfiable, we extend it by two clauses $(x_1 \vee x_2)$ and $(x_1 \vee x_3)$:

$$\begin{aligned} F_2 \quad = \quad & (\overline{x}_1 \vee \overline{x}_2) \wedge (x_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4) \wedge \\ & (x_2 \vee x_3) \wedge (\overline{x}_2 \vee \overline{x}_3)\,. \end{aligned} \tag{10.2}$$

We assume again that $x_1 = 1$ will be chosen first. Since this has already led to a contradiction when run without the two additional clauses (see above), it will lead to a contradiction again here. We do not present the details.

Now $x_1 = 0$ will be chosen, and UCP has to be called again. The first iteration will find the clauses $(\overline{x}_1 \vee \overline{x}_2)$ and $(\overline{x}_1 \vee \overline{x}_3)$ satisfied, hence they will be removed from the formula. In the clauses $(x_1 \vee x_2)$, $(x_1 \vee x_3)$, and $(x_1 \vee x_4)$, the literal x_1 is "false", hence the clauses will be reduced and moved to Γ_1. We obtain

$$\Gamma_0 = \emptyset,\ \Gamma_1 = \{(x_2), (x_3), (x_4)\},\ \Gamma_2 = \{(x_2 \vee x_3), (\overline{x}_2 \vee \overline{x}_3)\}\,.$$

In the next iteration, say, x_2 is chosen, hence we obtain

$$\Gamma_0 = \emptyset,\ \Gamma_1 = \{(x_3), (\overline{x}_3)(x_4)\},\ \Gamma_2 = \emptyset\,.$$

When treating either x_3 or $\overline{x}_3$, UCP notices the contradiction, i. e., it returns with a non-empty set, Γ_0. Hence, simple-2-SAT terminates by recognizing that the formula is not satisfiable. □

For 2-SAT, this algorithm exhibits a polynomial running time. For an unsatisfiable formula, the assignment of a variable may lead to assignments of *almost all*, i. e., $\mathcal{O}(n)$ other variables, before a contradiction is detected. Each time all $\mathcal{O}(m)$ clauses will be affected in this case. Since this may happen for all of the n variables, the worst-case running time of the 2-SAT algorithm is $\mathcal{O}(n^2 m)$.

The satisfiability of 2-SAT formulas can be checked faster, even in linear time [10]. The basic idea is to represent each clause by a pair of logical *implications*. Let us consider the clause $(l_1 \vee l_2)$, l_1, l_2 being two literals. We look for satisfying assignments. If the literal l_1 is "false", then literal l_2 must be true. Hence $l_1 = 0$ *implies* $l_2 = 1$. This can be written as $\overline{l}_1 \rightarrow l_2$, with $\rightarrow$ representing the logical implication operator. It can be defined by the following *truth table*

x_1	x_2	$x_1 \rightarrow x_2$
0	0	1
0	1	1
1	0	0
1	1	1

(10.3)

Note that an implication is only wrong, if something "false" is implied by something "true" (because logically one cannot deduce something that is wrong from something that is true). In the same way, for $(l_1 \vee l_2)$, $l_2 = 0$ *implies* $l_1 = 1$, i. e., we have $\overline{l}_2 \rightarrow l_1$. This means the clause $(l_1 \vee l_2)$ is equivalent to two implications, to the expression $I = (\overline{l}_1 \rightarrow l_2) \wedge (\overline{l}_2 \rightarrow l_1)$. You can also check this manually by evaluating the truth tables for $(l_1 \vee l_2)$ and for I for all four possible assignments of l_1, l_2, see Sec. 4.7 for the truth tables of the operators $\wedge$ and $\vee$.

In this way, each 2-CNF formula having n variables $x_1, \ldots, x_n$ and m clauses is equivalent to $2m$ implications. These can be represented as a directed graph $G = (V, E)$ with $2n$ vertices $V = x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n$. For each clause $(l_1 \vee l_2)$, E contains two directed edges $(\overline{l}_1, l_2)$ and $(\overline{l_2}, l_1)$. For example, for the clause $(x_3 \vee \overline{x}_5)$ the directed edges $(\overline{x}_3, \overline{x}_5)$ and (x_5, x_3) are contained in E. The graph G can be used to assign quickly all values, which result as implications from any assignment, i. e., when running UCP. Whenever a literal l is set to "true", one can follow immediately the edges in the graph and set also all literals l' to "true", which are accessible from l via directed paths. Additionally, $\overline{l}$ and all encountered literals $\overline{l'}$ are set to "false".

To understand how the fast method of deciding 2-SAT works, we realize that a 2-CNF formula F is satisfiable iff it does not contain any contradictions. A contradiction is equivalent to the existence of some variable x_i where $x_i = 1$ implies $x_i = 0$ *and* $x_i = 0$ implies $x_i = 1$ at the same time. Hence, F is not satisfiable iff in the corresponding graph G there is one variable x_i having a directed path from x_i to $\overline{x}_i$ and also a directed path from $\overline{x}_i$ to x_i. Recalling the definition of the *strongly connected component* (SCC) (see Sec. 3.1.1) this means that a 2-CNF formula is not satisfiable iff there is at least one variable x_i where x and $\overline{x}_i$ belong to the same SCC.

This implies a linear-time algorithm for deciding 2-SAT. One simply calculates the SCCs, which can be done in linear time, see Sec. 3.2.2. During the calculation, one keeps track of each vertex, in which SCC is. Finally, it is checked whether for some i, both x_i and $\overline{x}_i$ are

contained in the same SCC. If this is the case, the formula is not satisfiable, otherwise it is satisfiable.

For a SCC S, we introduce the *dual* SCC $\overline{S}$, which is obtained by complementing all vertices of S and reversing all edges within S. Note that an 2-CNF is not satisfiable iff there is a SCC S with $S = \overline{S}$.

If the formula is satisfiable, one can also easily find a satisfying assignment. In the following we speak of assignments to literals. Assigning the literal l to the value a means $x_i = a$ if $l = x_i$ and $x_i = \overline{a}$ if $l = \overline{x}_i$ First, we notice that all literals belonging to the same SCC must receive the same assignment. Furthermore, one has to consider the directed edges (S, S') *between* different strongly connected components S, S', i. e., the edges from E between vertices belonging to *different* SCCs. Note that the existence of an edge (S, S') implies that no edge (S', S) exists, because otherwise S, S' would have been merged to one big SCC. The edges between different SCCs imply a *topological order*. For edge (S, S') one says S *precedes* S'. Note that this order does exist, because there cannot be any loop. Otherwise all components along the loop would belong to the same component. Since an edge (S, S') has the meaning of the implication $S \to S'$, the literals of S' have to be assigned the value "true" if the literals of "S" are assigned the value "true". Hence, to avoid contradictions, one processes the SCCs in *reverse* topological order, always assigning the value "true" to yet unassigned components S (and "false" to $\overline{S}$). This is safe because, as we have seen, an implication $x_1 \to x_2$ is always "true" if x_2 is "true", no matter what value is assigned to x_1. After all SCCs have been assigned truth values, one assigns each literal the truth value of the SCC to which it belongs.

The linear-time algorithm for 2-SAT can be summarized as follows. Again Γ_2' contains all clauses of the 2-CNF formula F and the set U contains all variables. We denote by SCC(x) the strongly connected component of vertex x.

algorithm lin-2-SAT(Γ_2', U)
begin
  generate equivalent graph $G = (V, E)$ with
    $V = \{x, \overline{x} | x \in U\}$, $E = \{(\overline{l}_1, l_2), (\overline{l_2}, l_1) | (l_1 \vee l_2) \in \Gamma_2'\}$;
  calculate strongly connected components SCC$= \{S\}$ of G;
  for all $x \in U$
    if $\mathrm{SCC}(x) = \mathrm{SCC}(\overline{x})$ **then**
      print unsatisfiable; **stop**;
  initialize all SCC as *unassigned*;
  process SCCs S of G in reverse topological order **do**
    if S is *unassigned* **then**
      assign to S "true" and to $\overline{S}$ "false";
  for all $x \in U$ **do**
    assign x the value of SCC(x);
end

Example: Linear algorithm

We consider again the formula F_1 from Eq. (10.1). The resulting implication graph is shown in Fig. 10.1, the resulting SCCs are indicated by dashed lines.

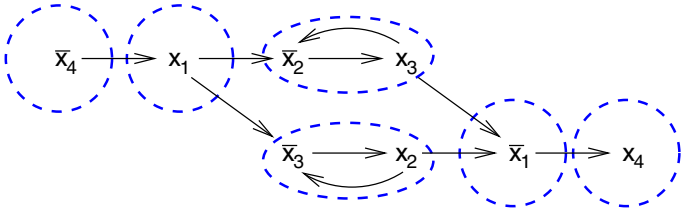


Figure 10.1: Implication graph for formula F_1 from Eq. (10.1). The resulting strongly connected components are indicated by the dashed lines.

There are two SCC with more than one member, $S_1 = \{\overline{x}_2, x_3\}$ and $\overline{S}_1 = \{x_2, \overline{x}_3\}$, with $S_1 \neq \overline{S}_1$, hence F_1 is satisfiable. To find a satisfying assignment, we treat the SCCs in reverse topological order. We start with $\{x_4\}$ and assign $x_4 = 1$ (hence SCC $\{\overline{x}_4\} = 0$). Next comes $\{\overline{x}_1\}$, i. e., $\overline{x}_1 = 1$, $(x_1 = 0)$. Now only S_1 and $\overline{S}_1$ are left, and we have a free choice, e. g., we can assign $\{x_2, \overline{x}_3\} = 1$, i. e., $x_2 = 1$, $x_3 = 0$.

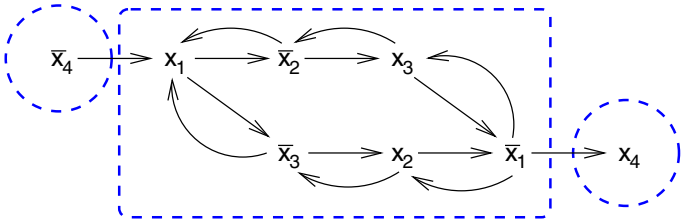


Figure 10.2: Implication graph for formula F_2 from Eq. (10.2). The resulting strongly connected components are indicated by the dashed lines.

The implication graph for formula F_2 from Eq. (10.2) is shown in Fig. 10.2. There is only one SCC S with more than one member. We have $S = \overline{S}$ (x_1 and $\overline{x}_1$, x_2 and $\overline{x}_2$, x_3 and $\overline{x}_3$ belong all to the same SCC),hence F_2 is not satisfiable.

□

10.1.2 Complete algorithms for K-SAT

The most obvious and slowest way to decide whether a K-SAT formula is satisfiable, is to try all possible 2^n assignments of the n variables $x_1, \ldots, x_n$. This does not use any structure

of the given formula and has obviously an exponential worst-case running time. Since 3-SAT is NP complete, one cannot hope to do much better. To improve the $\mathcal{O}(2^n)$ running time, assuming P≠NP, one can only look for algorithms which either run fast on typical instances of some ensemble, or which have a worst-case running time $\mathcal{O}(\omega^n)$ with $\omega < 2$. We now present one example for each case.

Davis–Putnam [13] introduced a SAT algorithm, which is is suitable for general K-SAT. It is based on *resolution*. This means that logical transformations of the given formula F (in K-CNF) are performed. In this case, a variable is selected, say x_i, and "factored out". This means F is written in the form $(A \vee x_i) \wedge (B \vee \overline{x}_i) \wedge R$, where A, B and R are free of x_i. A is a conjunction of all clauses containing x_i, with x_i deleted, B is a conjunction of all clauses containing $\overline{x}_i$, with $\overline{x}_i$ deleted, and R contains all clauses without variable x_i. F is only satisfiable iff $(A \vee B) \wedge R$ is satisfiable. This resolution process is continued, together with UCP, until a contraction occurs, or if the formula is satisfied. Note that the formula $(A \vee B) \wedge R$ must be written in K-CNF again, hence it may grow considerably in length. This often leads to an exponential growth of the memory used.

This problem is avoided by the algorithm of Davis–Logemann–Loveland [14], who replaced the resolution mechanism by *splitting*, i. e., a tree of all possible configurations is built. This works by again selecting at each node a variable x_i, but now creating two subtrees via assigning $x_i = 0$ ("left" subtree) and $x_i = 1$ ("right" subtree). Subtrees are omitted, if definitely no solution can be found. Hence, the Davis–Putnam–Logemann–Loveland (DPLL) algorithm is a branch-and-bound algorithm, see Sec. 6.3 for an introduction to branch-and-bound and its realization for the vertex-cover problem. The bound of the DPLL algorithm, allowing us to prune some subtrees, is realized by performing UCP after each assignment, and immediately returning, if a contradiction has been found. The details of the algorithm are as follows. Again a CNF formula is represented as a collection $\Gamma_0, \Gamma_1, \ldots, \Gamma_K$ of clauses and U denotes the set of unassigned variables. For K-SAT, initially $\Gamma_0, \ldots, \Gamma_{k-1}$ are empty, Γ_K contains all clauses, and U all variables.

algorithm DPLL($\Gamma_0, \Gamma_1, \ldots, \Gamma_k, U$)
begin
  UCP($\Gamma_0, \Gamma_1, \ldots, \Gamma_k, U$);
  if $\Gamma_0 \neq \emptyset$ **then** { unsatisfiable }
    return;
  if $\Gamma_0 = \Gamma_1 = \ldots = \Gamma_k = \emptyset$ **then**
    print satisfiable; **stop**;
  Let x a variable in U;
  DPLL($\Gamma_0, \Gamma_1 \cup \{x\}, \ldots, \Gamma_k, U$); {left subtree, try $x = 1$}
  DPLL($\Gamma_0, \Gamma_1 \cup \{\overline{x}\}, \ldots, \Gamma_k, U$); {right subtree, try $x = 0$}
end

Here we randomly choose the order of appearance of variables. At each branching point we always first assign the value "true", and then "false". Any sensible heuristic determining the order of the variables or choosing which value to assign first (e. g., choosing to assign "false" first, if there are more negated than non-negated variables) can speedup the algorithm.

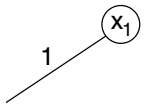
Example: DPLL

As an example to see how the DPLL algorithm works, we consider here the formula

$$F_3 = (x_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee x_4) \wedge (\overline{x}_1 \vee x_3 \vee x_4) \wedge (\overline{x}_2 \vee x_3 \vee \overline{x}_4) \wedge (\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4)\,. \tag{10.4}$$

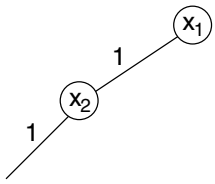
For the first call of DPLL, all clauses are contained in Γ_3, there are no unit clauses, hence UCP returns the formula unaffected. Therefore, no contraction has occurred, i. e., the algorithm starts to branch by assigning first, say, $x_1 = 1$. The algorithm calls the next level of DPLL. There, UCP finds that the clause $(x_1 \vee \overline{x}_2 \vee \overline{x}_4)$ is satisfied, it is removed. Two clauses, $(\overline{x}_1 \vee \overline{x}_3 \vee x_4)$ and $(\overline{x}_1 \vee x_3 \vee x_4)$ contain $\overline{x}_1$, hence they are reduced. This leads to the following situation, on the left we show the currently assigned variables and the sets Γ_i after the call to UCP has been finished in the first branching level; on the right, the current configuration tree:

$$\begin{aligned} &x_1 = 1 \\ &\Gamma_0 = \emptyset \\ &\Gamma_1 = \emptyset \\ &\Gamma_2 = \{(\overline{x}_3 \vee x_4), (x_3 \vee x_4)\} \\ &\Gamma_3 = \{(\overline{x}_2 \vee x_3 \vee \overline{x}_4), (\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4)\}\,. \end{aligned}$$



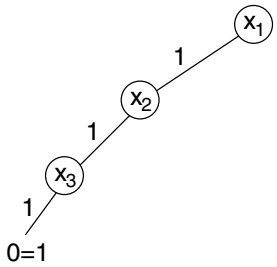
Again, first the left branch is taken, i. e., $x_2 = 1$ is assigned and DPLL is called. Since the literal x_2 does not appear in any clause, no clauses are removed. Only $\overline{x}_2$ is removed from the clauses $\{(\overline{x}_2 \vee x_3 \vee \overline{x}_4)$ and $(\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4)\}$, leading to the following situation:

$$\begin{aligned} &x_1 = 1, x_2 = 1 \\ &\Gamma_0 = \emptyset \\ &\Gamma_1 = \emptyset \\ &\Gamma_2 = \{(\overline{x}_3 \vee x_4), (x_3 \vee x_4), \\ &\qquad (x_3 \vee \overline{x}_4), (\overline{x}_3 \vee \overline{x}_4)\} \\ &\Gamma_3 = \emptyset\,. \end{aligned}$$



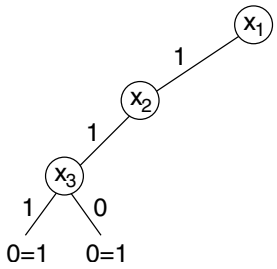
Next $x_3 = 1$ is assigned and the next level of the recursion is called. There, UCP removes the clauses $(x_3 \vee x_4)$ and $(x_3 \vee \overline{x}_4)$, while it removes $\overline{x}_3$ from $(\overline{x}_3 \vee x_4)$ and $(\overline{x}_3 \vee \overline{x}_4)$ yielding $\Gamma_1 = \{(x_4), (\overline{x}_4)\}$, resulting in an immediate contradiction detected by UCP. Hence we obtain ($0 = 1$ representing the contradiction)

$x_1 = 1, x_2 = 1, x_3 = 1$
$\Gamma_0 = \{()\}$
$\Gamma_1 = \emptyset$
$\Gamma_2 = \emptyset$
$\Gamma_3 = \emptyset\,.$



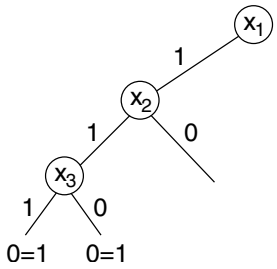
The algorithm backtracks one level. There $x_3 = 0$ is assigned and again DPLL called. UCP removes the clauses $(\overline{x}_3 \vee x_4)$ and $(\overline{x}_3 \vee \overline{x}_4)$, while it removes x_3 from $(x_3 \vee x_4)$ and $(x_3 \vee \overline{x}_4)\}$. Again $\Gamma_1 = \{(x_4), (\overline{x}_4)\}$ is obtained, resulting in a contradiction.

$x_1 = 1, x_2 = 1, x_3 = 0$
$\Gamma_0 = \{()\}$
$\Gamma_1 = \emptyset$
$\Gamma_2 = \emptyset$
$\Gamma_3 = \emptyset$



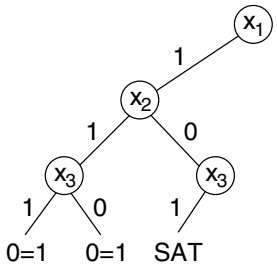
DPLL backtracks to the level where x_3 is assigned. Both values for x_3 have been tried, hence DPLL backtracks another level. Now $x_2 = 0$ is assigned and DPLL called. In the subsequent call to UCP, the clauses $(\overline{x}_2 \vee x_3 \vee \overline{x}_4)$ and $(\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4)$ are found to be satisfied, i. e., they are removed, resulting in

$x_1 = 1, x_2 = 0$
$\Gamma_0 = \emptyset$
$\Gamma_1 = \emptyset$
$\Gamma_2 = \{(\overline{x}_3 \vee x_4), (x_3 \vee x_4)\}$
$\Gamma_3 = \emptyset\,.$



Now $x_3 = 1$ is assigned and DPLL called. On the next level, UCP assigns $x_4 = 1$. No clauses are left, a satisfying assignment has been found. The algorithm terminates.

$x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1$
$\Gamma_0 = \emptyset$
$\Gamma_1 = \emptyset$
$\Gamma_2 = \emptyset$
$\Gamma_3 = \emptyset\,.$



□

Many variants of DPLL and other complete approaches exist [9,15]. Here we mention only, as an example, that for backtracking one does not need to follow the inverse order of the variables as they have been assigned, e. g., after assigning $x_1 = 1$ and then $x_2 = 1$, first x_1 might be changed. In this case one can use the knowledge of where the source of a contradiction is located, and change this assignment directly, this is called *non-chronological backtracking* [16, 17], leading to considerable speedup in some cases.

DPLL still has the worst-case running time $\mathcal{O}(2^n)$ found for the simple truth table. But when applied, e. g., to 2-SAT, the running time will be polynomial, because in this case DPLL becomes equivalent to the simple-2-SAT algorithm given above. Also, on ensembles of random K-SAT formulas, there are regions where DPLL runs polynomially. We will give a numerical example in Sec. 10.2.1.

We close this section, by presenting an algorithm proposed by Monien and Speckenmeyer [18], which runs, even in the worst case without UCP, faster than $\mathcal{O}(2^n)$, here we discuss the version for 3-SAT. The algorithm uses a divide-and-conquer approach and assigns variables in a controlled way. In the version we show here, it does not use UCP, which simplifies the analysis of the running time. The basic idea is to use the knowledge contained in the formula. At each step, one considers exactly *one* 3-literal clause $c = (l_1 \vee {}_2 \vee l_3)$. We can assume that there is such a clause with all three variables unassigned, otherwise, we have effectively a 2-SAT formula and we can use the linear-time algorithm. Since all clauses have to be satisfied for a satisfying assignment,c also has to be satisfied. This means that we can first try to assign the variable of l_1 such that $l_1 = 1$, this is one way to satisfy c. Then the process continues recursively. If this assignment has not been successful, i. e., does not lead to a satisfying assignment, we know $l_1 = 0$ must hold for a satisfying assignment and can try $l_2 = 1$, again continuing recursively. If this also fails, we know $l_1 = 0$, $l_2 = 0$ and only $l_3 = 1$ is left to satisfy c. The process is continued recursively a last time. If this fails again, we know that the formula is not satisfiable. We do not have to try $l_1 = 0, l_2 = 0, l_3 = 0$ because in this case already c is not satisfied. The algorithm can be summarized as follows: it receives as parameters the formula F and the current values of the variables (with, e. g., -1 meaning "not assigned") in $\mathbf{x} = x_1, \ldots, x_n$.

```
algorithm monien–speckenmeyer(F, x)
begin
  if all variables assigned then
    return F(x_1, ..., x_n);
  if there is a clause (l_1 ∨ l_2 ∨ l_3) with all three literals unassigned then
  begin
    l_1 := 1;
    if monien–speckenmeyer(F, x) = 1 then
      return 1;
    l_1 := 0; l_2 := 1;
    if monien–speckenmeyer(F, x) = 1 then
      return 1;
    l_1 := 0; l_2 := 0; l_3 := 1;
    if monien–speckenmeyer(F, x) = 1 then
      return 1;
    return 0;
  end
  else
    return value of F evaluated with lin-2-SAT;
end
```

Note that when assigning to a literal, e. g., $l_1 := 1$, we again mean that we assign the corresponding variable a value, such that the evaluation of the literal gives the desired result.

In the worst case, the lin-2-SAT is never called. In this case, we can easily calculate the running time $T(n)$ of the algorithm with n being the number of yet unassigned variables. Omitting constants leading only to sub-dominant polynomial factors, we get the recurrence equation (see Chap. 2),

$$T(n) = \begin{cases} 1 & \text{for } n = 0, 1, 2 \\ T(n-1) + T(n-2) + T(n-3) & \text{for } n > 2 \end{cases} . \tag{10.5}$$

It is clear that the worst-case running time will be exponential. Hence, we can use as a guess $T(n) = \alpha^n$. Plugging this into Eq. (10.5) we obtain $\alpha^n = \alpha^{n-1} + \alpha^{n-2} + \alpha^{n-3}$, i. e., $\alpha^3 = \alpha^2 + \alpha + 1$. The solution is $\alpha \approx 1.839$, hence the worst-case running time of the Monien–Speckenmeyer algorithm is $\mathcal{O}(1.839^n)$, slightly better than $\mathcal{O}(2^n)$. More recent deterministic algorithms [19] have a worst-case running time of $\mathcal{O}(1.476^n)$, and $\mathcal{O}(1.481^n)$ [20, 21]. One can do much better using simple stochastic algorithms, which are discussed in the following section.

10.1.3 Stochastic algorithms

Stochastic algorithms explore the assignment space (=configuration space) in a random fashion, looking for a solution, i. e., a satisfying assignment. This is similar to the application of Monte Carlo methods for the vertex-cover problem, see Sec. 6.6. Hence, stochastic algorithms can prove only satisfiability, they are not able to prove unsatisfiability. This means they are *incomplete*algorithms. Their real power comes into play, when applied to formulas that are (almost surely) known to be satisfiable. This is true in particular for random SAT formulas in the satisfying region of the phase diagram, see the next two sections. For those applications, stochastic methods are very often faster than complete algorithms. Hence, we assume in this section that a satisfying assignment $\mathbf{x}^* = x_1^*, \ldots, x_n^*$ exists. In addition, if one is able to calculate a bound on the probability that the stochastic search finds the solution, one can run the program for so long that the probability of missing an existing solution becomes very small, e. g., e^{-20}. In this case, if the algorithm does not find a solution, one can be quite sure that the formula is unsatisfiable.

We first introduce *RandomWalkSAT*. Then we extend it by using the concept of random *restarts*. For this case, we will derive a probabilistic bound on the worst-case running time, allowing us to use the algorithm also in the case where no solution exists. Other variants of stochastic SAT algorithms and related heuristics have been discussed in recent reviews [9,22].

The basic idea of RandomWalkSAT [23] is very simple. One starts with one randomly selected assignment, which is updated iteratively. In each algorithmic step one selects randomly one unsatisfied clause c, and one variable x_i contained in c. The value of this variable becomes negated ($x_i := \overline{x}_i$). Thus, clause c will becomes satisfied, some additional clauses might become "true" as well, while other clauses might turn "false". In the case when all clauses are satisfied after such an update, the algorithm stops. This iteration is repeated, at most, $t_{\max}$ times, if this running-time threshold is reached, the algorithm exits without having found a satisfying assignment to the Boolean variables.

The algorithm can be summarized as follows:

algorithm RandomWalkSAT($F, \mathbf{x}, t_{\max}$)
begin
  for $t := 1, \ldots, t_{\max}$ **do**
  begin
    choose randomly an unsatisfied clause c;
    select on variable x_i randomly in c;
    flip variable: $x_i := \overline{x}_i$;
    if satisfying assignment of F has been found **then**
      print satisfiable; **stop**;
  end
end

The RandomWalkSAT algorithm performs a random walk in assignment space, see Fig. 10.3. To understand the algorithm better and to analyze its running time, it is useful to consider the

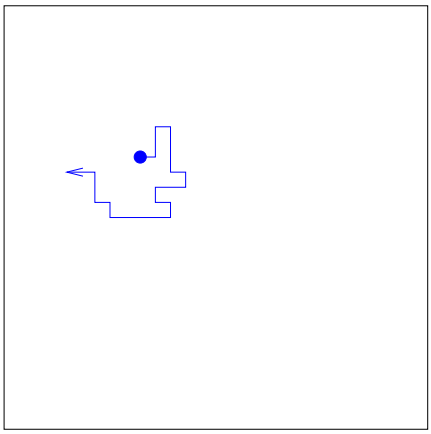


Figure 10.3: Schematic representation of the RandomWalkSAT. One starts at a random starting configuration (circle). The box represents the n-dimensional assignment space. Iteratively an unsatisfied clause is selected and one randomly chosen variable is flipped, leading to a random walk in assignment space.

Hamming distance $j \equiv d(\mathbf{x}^*, \mathbf{x})$ of the current assignment $\mathbf{x}$ to the assumed solution $\mathbf{x}^*$. One can depict the random-walk process by a one-dimensional Markov chain $y(t)$ (see Sec. 6.6.2), where the states y are numbered according to the Hamming distance j, i. e., $y = 0, 1, 2, \ldots, n$, see Fig. 10.4. We denote by $r_{j,j-1}$ the transition probability from state j to state $j-1$, i. e., the probability that the current transition decreases the Hamming distance to the solution by one unit. The Markov state 0, corresponding to the case when the solution has been found, is *absorbing*, because the algorithm stops after the solution has been found. This means, there is no transition leaving state 0. There might be other satisfying assignments, hence, the algorithm might stop being in other states. To facilitate the analysis, we assume that the Markov process does *not* stop, unless it has reached state 0. In this way, when asking for the probability that RandomWalkSAT finds a satisfying solution, i. e., for the *success probability*, the result is bounded from below by the probability that the Markov process studied here reaches state 0.

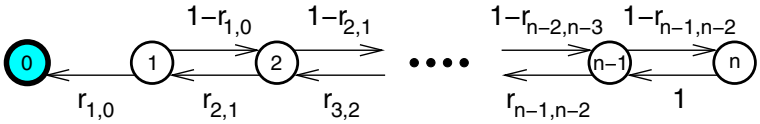


Figure 10.4: The RandomWalkSAT can be seen as a one-dimensional walk in Hamming space. The label of the configuration (circles) denote the distance $d(\mathbf{x}^*, \mathbf{x})$ to the solution $\mathbf{x}^*$. The transition rates are unknown.

Unfortunately, we do not know the transition probabilities $q_{j,j-1}$, because when picking an unsatisfied clause, there might be several variables occurring in the clause, which have values different from the solution $\mathbf{x}^*$. Nevertheless, we know that *at least one* of the K variables has a value different from the solution. Hence, we obtain $q_{j,j-1} \geq 1/K$ for the transition

probabilities leading *towards* the solution. Since we have, in any case, restricted our analysis to finding a lower bound for the success probability, we can perform the analysis of the Markov chain with transition probabilities $q^b_{j,j-1} \mapsto 1/K$.

The state n, corresponding to $\mathbf{x} = \overline{\mathbf{x}^*}$, is a *reflecting* state, hence the transition probability $q_{n,n-1} = 1$. To facilitate the analysis even more, we instead assume that the states extend to infinity to the right, again with the transition probabilities $q^b_{j,j-1} \equiv 1/K$ and $q^b_{j,j+1} = 1 - q^*_{j,j-1}$. This still provides an (even) lower bound on the probability of finding a solution.

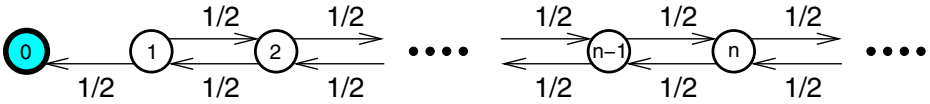


Figure 10.5: The WalkSAT can be seen as a one-dimensional walk in Hamming space. The labels of the configuration (circles) denote the distance $d(\mathbf{x}^*, \mathbf{x})$ to the solution $\mathbf{x}^*$. To obtain a lower bound on the success probability, the transition probabilities leading towards the solution can be bounded by $1/2$ for 2-SAT and it can be assumed that the right boundary of the Markov chain is infinite.

For 2-SAT, we have $q^b_{i,j-1} = 1/2$. Therefore, the Markov process describes a one-dimensional unbiased random walk along the chain of states. The walker starts at some state j_0 and in each step moves left or right, each with probability $1/2$, see Fig. 10.5. The one-dimensional random walk is a standard model in graduate text books on statistical mechanics [24], hence we do not go into the details here. We only mention that the standard deviation of the distribution of positions along the chain grows with the square-root with the number $t_{\max}$ of steps. Since the typical distance from the target state 0 grows linearly with n, the running time $t_{\max}$ has to increase quadratically with n, to reach state 0 with finite probability. This means that the RandomWalkSAT has a probabilistic worst-case running time of $\mathcal{O}(n^2)$, i. e., it is polynomial. Since there is already a linear-time deterministic algorithm, the RandomWalkSAT has no practical meaning for 2-SAT.

Instead, the algorithm works very efficiently for K-SAT with $K > 2$. Usually, one does not apply the simplest RandomWalkSAT algorithm. It is combined with some heuristics, in general, completely random and heuristic steps are alternated stochastically. The most famous greedy heuristics [25] is included in WalkSAT, one of the most efficient implementations of SLS for SAT. In a greedy step, one still randomly selects an unsatisfied clause, but inside this clause the variable is flipped which causes the minimal number of previously satisfied clauses to be violated (the so-called break-count). The probability of selecting a greedy step is optimized with respect to the probability of RandomWalkSAT steps. A recent review of different SLS methods can be found in Refs [9, 22]. Even with the most clever heuristics, one still needs, in the worst-case, an exponential number $t_{\max} \in \mathcal{O}(\omega^n)$ of steps to find a solution with high probability. A small value $\omega < 2$ provides a speedup compared with many deterministic algorithms. Instead of providing an analysis for the RandomWalkSAT, which is often impossible when the RandomWalkSAT is used in conjunction with efficient heuristics for choosing variables, we now study an extended version of the RandomWalkSAT. It, together

with its latest variants, exhibit the best probabilistic bound on the worst-case running time known so far, better than all deterministic algorithms.

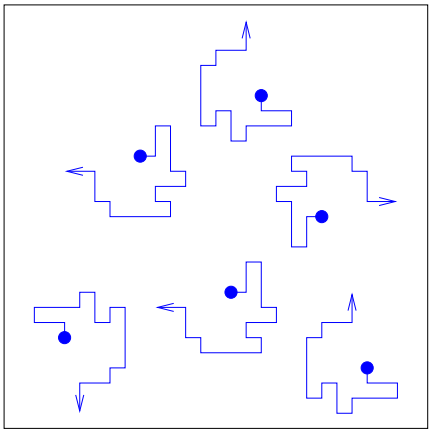


Figure 10.6: The RandomWalkSAT with restarts. One starts at different random starting configuration (circles) in the n-dimensional assignment space (box). Each time, iteratively, an unsatisfied clause is selected and one randomly chosen variable is flipped, leading to many random walks in assignment space, one after the other.

This extension increasing the efficiency of the RandomWalkSAT algorithm is based on the concept of restarts [26, 27]. Already in the case of the simple vertex-cover algorithm in Sec. 8.3 we have seen that this simple modification may lead to an exponential speedup of the computational time. The main idea is to perform *several* independent RandomWalkSAT searches from several random starting assignments, one search after the other, see Fig. 10.6. Each stochastic search is performed *only* for a restricted number $t_{\max} \sim \mathcal{O}(n)$ of steps. Here we use $t_{\max} = 3n$, for simplicity of analysis as we will see. This means that if no solution has been found during this time, we repeat from a new randomly chosen assignment, i. e., we perform a restart. The maximum number of restarts is denoted by $t(n)$. The basic frame of the algorithm reads as follows:

algorithm restart(F)
begin
  for $t := 1, \ldots, t(n)$ **do**
  begin
    choose random assignment $\mathbf{x} = x_1, \ldots, x_n$;
    RandomWalkSAT($F, \mathbf{x}, 3n$);
    if satisfying assignment of F has been found **then**
      stop;
  end
end

How should we choose $t(n)$? Let $p(n)$ be the probability that WalkSAT finds a satisfying assignment during one run of RandomWalkSAT (using at most $3n$ steps), i. e., the *success probability*. The probability that we do *not* find an existing satisfying assignment within one run is then $(1-p(n))$. The probability that we do not find a solution within $t(n)$ runs is hence $p_{\text{fail}} \equiv (1-p(n))^{t(n)}$. Since $\log x \leq x-1$, we obtain $(1-p(n))^{t(n)} \leq e^{-p(n)t(n)}$. Hence, if we choose $t(n) = c/p(n)$, with, e. g., $c = 20$, we have $p_{\text{fail}} \leq e^{-20}$ which should be safe.

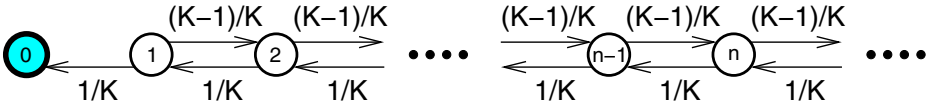


Figure 10.7: The WalkSAT can be seen as a one-dimensional walk in Hamming space. The label of the configuration (circles) denote the distance $d(\mathbf{x}^*, \mathbf{x})$ to the solution $\mathbf{x}^*$. To obtain an lower bound on the success probability, the transition probabilities leading towards the solution can be bounded by $1/K$ for K-SAT and it can be assumed that the right boundary is infinite.

Hence, we have to calculate [26,27] the success probability $p(n)$ for RandomWalkSAT, to obtain $t(n)$. To obtain a lower bound on $p(n)$, leading to an upper bound on $t(n)$, we study the Markov chain shown in Fig. 10.7. First, we assume that all initial assignments are equiprobable. Hence, the probability of starting in state j ($j = 0, 1, \ldots, n$) of the corresponding Markov chain follows a binomial distribution

$$P(y(0) = j) = 2^{-n} \binom{n}{j} . \tag{10.6}$$

To obtain the desired result, we have first to calculate the probability q_j (or a lower bound), which we define as the probability that the Markov chain reaches state 0 within $3n$ steps when starting at state j. Then we take the average of q_j over $P(y(0) = j)$. Since we expect an exponential worst-case running time, we ignore polynomial factors in n for the following calculation.

First, we calculate a lower bound on q_j. The random walk starting at state j can include i steps to the right (R), i. e., away from the target. Hence, to reach state 0, one must perform $i + j$ steps to the left (L). This means, in total, the walk consists of $j + 2i$ steps. Note that we again use infinite boundaries on the right. We denote by $q(i, j)$ the probability that we start at state j, move i steps to the right and $i + j$ steps to the left (with $i + 2j \leq 3n$) and reach state 0 the first time, exactly after $i + 2j$ steps. This means that we can perform a sum over the different possible values of i to obtain q_j:

$$\begin{aligned} q_j &= \sum_{2i+j \leq 3n} q(i,j) \\ &\geq \sum_{i=0}^{j} q(i,j) . \end{aligned} \tag{10.7}$$

The probability for one sequence of $i + j$ L and i R steps is $(1/K)^{i+j}((K-1)/K)^i$. If we denote by $N(i + j, i)$ the number of feasible sequences of $i + j$ symbols L and i symbols R,

we obtain

$$q(i,j) = N(i+j,j)\left(\frac{1}{K}\right)^{i+j}\left(\frac{K-1}{K}\right)^{i}. \tag{10.8}$$

The total number of these L/R sequences is $\binom{i+2j}{i}$, but we should not include those sequences, where the random walk arrives at state 0 before the last step. For example, when studying the case $j = 2, i = 1$, we have the following sequences

$$\left.\begin{array}{l} RLLL \\ LRLL \\ LLRL \\ LLLR \end{array}\right\} \quad \text{not feasible} \tag{10.9}$$

To find the number of feasible sequences of L/R steps, we use the *ballot theorem* [28, 29]:

> *In a ballot, candidate A scores a votes, candidate B scores b votes, with $a \geq b$. Assuming that all orderings are equally likely, the probability $P_{\mathrm{A}>\mathrm{B}}(a,b)$ that A is always ahead throughout the counting is $(a-b)/(a+b)$.*

Proof:

To prove the ballot theorem, we observe that the probability is one for $b = 0$ and it is zero for $a = b$, as stated by the formula. The further proof is by mathematical induction. We assume that $a > b$ and $b > 0$ holds and that the formula is true for the two cases $a' = a-1, b' = b$ and $a' = a, b' = b-1$. The probability that the last vote counted is for A is $a/(a+b)$, while the probability that the last vote counted is for B is $b/(a+b)$. Since it is sure that A is ahead in the last step of the counting ($a > b$), the probability that A is always ahead is

$$\begin{aligned} P_{A>B}(a,b) &= \frac{a}{a+b}P_{A>B}(a-1,b) + \frac{b}{a+b}P_{A>B}(a,b-1) \\ &= \frac{a}{a+b}\frac{(a-1)-b}{(a-1)+b} + \frac{b}{a+b}\frac{a-(b-1)}{a+(b-1)} \\ &= \frac{a^2-a-ab+ab-b^2+b}{(a+b)(a+b-1)} \\ &= \frac{(a+b-1)(a-b)}{(a+b)(a+b-1)} = \frac{a-b}{a+b} \end{aligned}$$

QED

The ballot theorem applies to our case. We see this immediately, when looking at the sequences in Eq. (10.9). Only those sequences are feasible, where, when read *in inverse order* from the right to left, the number of L steps is always ahead of the number of R steps, otherwise the random walk would have visited state 0 before the last step. Hence, the ballot theorem tells us that only the fraction $P_{L>R}(i+j,i) = j/(2i+j)$ of the possible $\binom{2i+j}{j}$

sequences is feasible. We obtain, from Eq. (10.8), and using $j/(2i+j) \geq 1/3$

$$\begin{aligned} q(i,j) &= \binom{2i+j}{j} \frac{j}{2i+j} \left(\frac{1}{K}\right)^{i+j} \left(\frac{K-1}{K}\right)^{i} \\ &\geq \frac{1}{3}\binom{2i+j}{j} \left(\frac{1}{K}\right)^{i+j} \left(\frac{K-1}{K}\right)^{i} . \end{aligned} \tag{10.10}$$

To simplify this expression, we apply Stirling's formula $N! \approx (N/e)^N$, which holds always up to a polynomial factor. In the general case we have

$$\binom{N}{L} = \frac{N!}{L!(N-L)!} \approx \frac{(N/e)^N}{(L/e)^L((N-L)/e)^{N-L}} = \left(\frac{N}{L}\right)^L \left(\frac{N}{N-L}\right)^{N-L} . \tag{10.11}$$

Applying this approximation to Eq. (10.10), and defining α by $i = \alpha j$ ($\alpha \in [0,1]$), we obtain

$$q(i,j) \geq \left[\left(\frac{1+2\alpha}{\alpha}\right)^{\alpha} \left(\frac{1+2\alpha}{1+\alpha}\right)^{1+\alpha} \left(\frac{1}{K}\right)^{1+\alpha} \left(\frac{K-1}{K}\right)^{\alpha} \right]^j . \tag{10.12}$$

Now, we bound the sum over i of $q(i,j)$ in Eq. (10.7) by its largest summand. The largest contribution can be obtained by calculating the derivative of the right-hand side of Eq. (10.12) with respect to α and setting it equal to zero. Using $z^z = \exp(z \ln z)$ and the fact that $\exp(z) > 0 \, \forall z$, one easily obtains $\alpha = 1/(k-2)$. Inserting this into Eqs (10.7) and (10.12) yields

$$q_i \geq \frac{1}{3} \left(\frac{1}{K-1}\right)^j . \tag{10.13}$$

As already indicated, to obtain a lower bound on $p(n)$, we average over the different initial states $j = y(0)$ via the binomial distribution $P(y(0) = j)$ given in Eq. (10.6). Using the binomial theorem $(s+t)^n = \sum_{j=0}^{n} \binom{n}{j} s^{n-j} t^j$, we obtain

$$\begin{aligned} p(n) &= 2^{-n} \sum_{j=0}^{n} \binom{n}{j} q_j \\ &\geq \frac{1}{3} 2^{-n} \sum_{j=0}^{n} \binom{n}{j} \left(\frac{1}{K-1}\right)^j \\ &= \frac{1}{3} \left(\frac{K}{2(K-1)}\right)^n . \end{aligned} \tag{10.14}$$

Thus, since $t(n) \sim 1/p(n)$, we obtain the final result for the required number of iterations $t(n)$ as (within a polynomial factor)

$$t(n) \in O\left(\left[\frac{2(K-1)}{K} \right]^n \right) . \tag{10.15}$$

Note that the calculation is valid only for $K > 2$, because $\alpha = 1/(K-2)$ was used. Nevertheless, a refined analysis [27] shows that the result is true for $K > 1$, and that the polynomial factor is exactly $2/3$, i. e., independent of n. Therefore, the running time is linear for 2-SAT, as for the fastest deterministic 2-SAT algorithm explained above.

For $K = 3$, Eq. (10.15) results in a worst-case running time of $\mathcal{O}(1.3334^n)$. A recent variant [30], where the initial assignments are not uniformly guessed, leads to the improvement $\mathcal{O}(1.3303^n)$. Other modifications lead to even further slight improvements resulting in worst-case running times of $\mathcal{O}(1.3302^n)$ [31] or $\mathcal{O}(1.3290^n)$ [32].

All these running times describe the worst-case behavior, i. e., they are upper bounds on the asymptotic running time. As we have seen for vertex cover, there are random ensembles of instances, where in some regions of the parameter space the instances are *typically* easy to solve. The first problem where this phase transition has been found, was the random 3-SAT model. This led to the application of concepts and methods from statistical physics, as discussed in this book. We discuss in the following sections this phase transition for random 3-SAT and the typical behavior of the RandomWalkSAT algorithm in this ensemble.

10.2 Phase transitions in random K-SAT

As already mentioned at the beginning of this chapter, K-SAT is the first combinatorial decision problem, for which a phase transition from a solvable to a non-solvable phase, connected to an easy–hard transition, was discovered [1, 2].

This was done in the framework of the development of a typical-case complexity theory using randomized problem instances, as discussed in Sec. 4.9, i. e., for *random K-SAT*. The latter is defined as a randomly generated version of K-SAT. Let us briefly summarize the definition:

Random 3-SAT is defined over a set of N Boolean variables $\{x_i\}_{i=1,\ldots,N}$, or, equivalently, over the $2N$ literals $\{x_i, \overline{x}_i\}_{i=1,\ldots,N}$. These are restricted by $M = \alpha N$ different K-clauses $\{C_\mu\}_{\mu=1,\ldots,M}$, each one consisting of a disjunction of exactly K literals, e. g., $C_\mu = (x_i \vee \overline{x}_j \vee x_k)$ for $K = 3$. The particularity of *random* K-SAT is that these *literals are selected randomly and independently* from the set of all $2N$ literals. In a random K-SAT formula F, these clauses are connected by logical AND operations.

$$F = \bigwedge_{\mu=1}^{M} C_\mu \; . \tag{10.16}$$

The formula is *satisfied*, if it evaluates to 1. This means that *all* M clauses have to be *satisfied simultaneously*. The relevant control parameter for the formulas generated in this way is the clauses-to-variables ratio $\alpha = M/N$. Note the close analogy of this ratio to the average vertex degree in random graphs. It is therefore not surprising that many of the phenomena described below have already been in previous chapters of this book, but in the context of the vertex-cover problem instead of SAT.

10.2.1 Numerical results

The formulas generated in this way are random, so also their satisfiability is a random variable. A first interesting quantity is therefore the probability that a random K-SAT formula is satisfiable, as a function of α, for fixed number N of Boolean variables. It is obvious that this probability will be close to one for small α, because there are only few constraints which are unlikely to induce a contradiction. For fixed K, the probability, however, is a monotonously decreasing function of α, since more and more constraints are added.

Measuring this quantity numerically, a surprising result was found. The decrease from probabilities close to one to those close to zero is not given by a slowly varying function of α, but it is characterized by a sharp drop in a limited α-region which is, e. g., for $K = 3$, situated close to $\alpha \simeq 4.2 \ldots 4.3$ [1,2], cf. also Fig. 10.8. This drop becomes sharper and sharper if the number of variables is increased. Finite-size scaling indicates that it tends eventually to a step function in the thermodynamic limit $N \to \infty$. Formulated in physical language, there exists a *phase transition* at a critical point $\alpha_c(K = 3) \simeq 4.26$. For $\alpha < \alpha_c(K)$, the randomly generated formula is *almost surely* satisfiable, i. e., the probability of being satisfiable tends to one in the large-N limit. For $\alpha > \alpha_c(K)$, on the other hand, the formula becomes almost surely unsatisfiable, i. e., it contains unresolvable contradictions. This transition is called the SAT/UNSAT-transition.

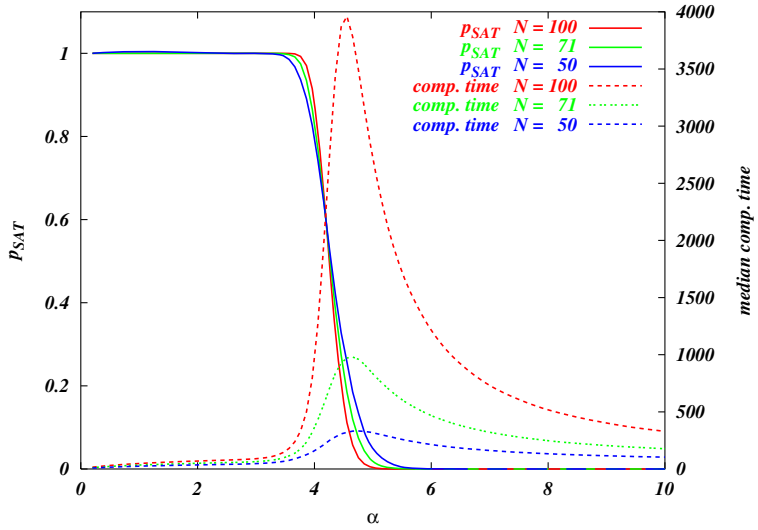


Figure 10.8: The 3-SAT/UNSAT transition. The figure shows the probability that a randomly generated 3-SAT-formula is satisfiable, as a function of the clauses-to-variable ratio α, and the median resolution time required by the DPLL algorithm. One can clearly see the probability drop close to $\alpha_c \simeq 4.26$ which sharpens with increasing variable number N. The median resolution time shows a pronounced exponential maximum close to α_c.

Even more interestingly for computer scientists, this phase transition is connected to a characteristic resolution time pattern, referred to as the *easy–hard–easy transition*. Resolution times are again measured as the number of algorithmic steps to obtain computer- and implementation-independent results. The average solution time is dominated by exponentially rare events requiring exponentially long running times. Since we are interested in measuring the most probable, or *typical* running time,the median solution time is considered here.

This median solution time, plotted again as a function of α, shows an interesting behavior. As can be seen in Fig. 10.8, it is polynomial for small α. Exponential solution times appear only beyond some algorithm-dependent threshold α_d which is located well inside the satisfiable phase, i. e., $\alpha_d < \alpha_c$. Directly at the phase transition point α_c, a pronounced exponential maximum of the required running time is found, for larger α the instances become again simpler to solve, even if the resolution time still grows exponentially with N. The hardest to solve formulas are thus found at the phase boundary, they are said to be critically constrained. A simple intuitive explanation can be given here. For small α, only a few constraints are present. So there are many solutions, and one of them can easily be found. This task becomes harder and harder if the number of constraints grows, and the number of solutions goes down. For large α, i. e., inside the unsatisfiable phase, the structures leading to logical contradictions in the formula become smaller for growing α. They are therefore more easily identified, and the unsatisfiability of a formula is proved more easily.

10.2.2 Rigorous mathematical results

The first and most fundamental mathematical question concerns the very existence of this phase transition. So far, we have only seen numerical evidence from small 3-SAT formulas. Does the conjectured sharp SAT/UNSAT threshold $\alpha_c(K)$ exist for every K?

A partial answer was given some years ago by Friedgut [33]. He proved that the *width* of the drop in probability, in fact tends to zero, and in the large-N limit, the transition becomes sharp. He was, however, not able to show the convergence to a well-defined asymptotic threshold. Theoretically, the position of the shrinking transition window can remain a non-trivial function of the number N of Boolean variables. Below we will see that the asymptotic position of the transition is restricted by rigorously established lower and upper bounds. A non-trivial dependence of the transition on N can therefore only be realized by an oscillating function which is bounded from above as well as from below. It is widely accepted that this scenario is extremely improbable and we will therefore assume the existence of well-defined asymptotic SAT/UNSAT thresholds for arbitrary clause length K, but the mathematical proof is still missing.

Upper bounds

Most upper bounds on the SAT/UNSAT threshold are based on refinements of the first-moment method [34–36]. As already explained in Sec. 7.2, the first moment bound is based on the

simple inequality bounding the probability that a random K-SAT formula of N variables and $M = \alpha N$ clauses is satisfiable:

$$Prob(F \text{ is sat.}) \leq \overline{\mathcal{N}_{sat}(F)} \tag{10.17}$$

where $\mathcal{N}_{sat}(F)$ denotes the number of satisfying assignments, and the overbar denotes the average over the random 3-SAT ensemble at fixed N and α. The right-hand side is easily evaluated. There are 2^N assignments to the N Boolean variables. Each clause, however, forbids 1 out of 2^K configurations of the K variables forming the clause, the other $2^K - 1$ configurations satisfy the clause. Since clauses are drawn independently, we find

$$\overline{\mathcal{N}_{sat}(F)} = 2^N \left(\frac{2^K - 1}{2^K}\right)^{\alpha N} . \tag{10.18}$$

This quantity depends exponentially on N. As long as it is increasing, the inequality (10.17) is trivial. If it, however, goes exponentially to zero, we also know that the probability of finding a satisfiable formula goes to zero. We are above the SAT/UNSAT threshold. The upper bound for the latter, results from setting the last expression to one, i. e.,

$$\alpha_{1st}(K) = -\frac{\ln 2}{\ln(1 - 2^{-K})} . \tag{10.19}$$

For $K = 3$ this results in $\alpha_c(3) < 5.191$, which is relatively far from the numerical value reported above. Asymptotically, for $K \gg 1$, the expression can be simplified by splitting $1/(\ln(1-x))$ for $x = 2^{-K} \to 0$ ($K \to \infty$) into a diverging and a non-diverging part $\frac{1}{x}\frac{x}{\ln(1-x)}$ and expanding the non-diverging part around $x = 0$ as $\frac{x}{\ln(1-x)} \approx -1 + x/2 + \mathcal{O}(x^2)$, which results in

$$\alpha_{1st}(K) = \left(2^K - \frac{1}{2}\right) \ln 2 + \mathcal{O}(2^{-K}) . \tag{10.20}$$

Refinements of the first-moment method bring the result much closer to the numerical value, the closest result [36] is $\alpha_c(3) < 4.51$. For the asymptotic behavior, the effect is less impressive, only the constant term is lower by 1/2. The best asymptotic upper bound [34] so far is

$$\alpha_{ub}(K) = \left(2^K - \frac{1}{2}\right) \ln 2 - \frac{1}{2} + \mathcal{O}(2^{-K}) . \tag{10.21}$$

Lower bounds

Lower bounds have been found mainly by using two basic methods.

The first one, already discussed several times in this book, consists in analyzing heuristic algorithms which explicitly construct solutions [37–39]. If such an algorithm can be shown to

find a satisfying assignment at some value of α with some positive probability (more precisely, it must be bounded away from zero), we know by the sharpness of the transition that this α forms a lower bound.

The best bound for 3-SAT which was constructed in this way is 3.42 [40]. For $K \gg 1$, all so far analyzable algorithms lead to the same dominant behavior of the asymptotic algorithmic lower bound,

$$\alpha_{alg}(K) = \frac{2^K}{K}(\ln K + \mathcal{O}(\ln \ln K)) \; . \tag{10.22}$$

Compared with Eq. (10.21), a factor of order $\mathcal{O}(K/\ln K)$ growing with K appears, the two bounds together are not able to pin the asymptotic behavior of the SAT/UNSAT threshold.

The second basic method for finding lower bounds is the so-called second-moment method. It is based on the inequality

$$\text{Prob}(X > 0) \geq \frac{\overline{X}^2}{\overline{X^2}} \tag{10.23}$$

which is valid for any random variable taking only non-negative integer values: Let p_i be the probability of obtaining value $i \geq 0$, then we have

$$\begin{aligned} \overline{X}^2 &= \left(\sum_{i>0} p_i i\right)^2 = \sum_{i,j>0} p_i p_j ij = 2 \sum_{0<j<i} p_i p_j ij + \sum_i p_i^2 i^2 \\ &\leq 2 \sum_{0<j<i} p_i p_j ii + \sum_i p_i^2 i^2 = \sum_{i,j>0} p_i p_j i^2 \\ &= \left(\sum_{j>0} p_j\right)\left(\sum_{i>0} p_i i^2\right) = \text{Prob}(X > 0)\overline{X^2}. \end{aligned} \tag{10.24}$$

An example for a possible choice of X is the number $\mathcal{N}(F)$ of solutions of a randomly generated formula F. Unfortunately, this example produces only an exponentially small lower bound for $Prob(\mathcal{N}(F) > 0)$, i. e., for the probability that F is satisfiable. Better examples, however, allow us to produce non-trivial lower bounds for the SAT/UNSAT threshold $\alpha_c(K)$ [41]. For small K they are worse than the best algorithmic bounds. Asymptotically one finds, however,

$$\alpha_{lb}(K) = 2^K \; \ln 2 - \left(\frac{K+1}{2}\ln 2 + 1 + o(1)\right) \tag{10.25}$$

with $o(1) \to 0$ for $K \to \infty$. The leading term coincides with the first-moment upper bound. Consequently, the latter describes correctly the asymptotically dominant behavior of $\alpha_c(K)$. It seems surprising that the results from all analyzable algorithms are so far away from this value. The statistical-mechanics analysis below will deliver a conjectured reason for this gap.

Note that the first- and second-moment methods have some similarity to the replica trick. It allows us to obtain some information on the distribution of a random variable from its integer moments. It would be interesting to extend this mathematically rigorous approach also to higher moments, but so far no further insight had been gained in this direction.

10.2.3 Statistical-mechanics results

More detailed results can be obtained using statistical physics. The existence of a phase transition in random 3-SAT obviously provides a temptation to apply tools developed in the field of statistical physics of disordered systems, like the replica trick or the cavity method. The analysis of random 3-SAT is based on a representation in terms of a diluted spin-glass model. [3] Boolean variables $x_i = 0, 1$ are mapped to Ising spins $S_i = 2x_i - 1$. The clauses are uniquely represented by the matrix

$$c_{\mu,i} = \begin{cases} +1 & \text{if } x_i \in C_\mu \\ -1 & \text{if } \overline{x}_i \in C_\mu \\ 0 & \text{else.} \end{cases} \tag{10.26}$$

The Hamiltonian counts the number of unsatisfied clauses,

$$\mathcal{H} = \sum_{\mu=1}^{\alpha N} \delta_{-K,\sum_i c_{\mu,i} S_i} \ . \tag{10.27}$$

For $K = 3$, it can easily be rewritten as

$$\mathcal{H} = \frac{\alpha}{8} N - \sum_{i=1}^{N} H_i S_i - \sum_{i<j} T_{ij} S_i S_j - \sum_{i<j<k} J_{ijk} S_i S_j S_k \ . \tag{10.28}$$

This Hamiltonian contains random local fields, two- and three-spin-interactions, representing the quenched disorder of the model. Their values are given by the random choice of the clauses,

$$\begin{aligned} H_i &= \frac{1}{8} \sum_\mu c_{\mu,i} \\ T_{ij} &= -\frac{1}{8} \sum_\mu c_{\mu,i} c_{\mu,j} \\ J_{ijk} &= \frac{1}{8} \sum_\mu c_{\mu,i} c_{\mu,j} c_{\mu,k} \ . \end{aligned} \tag{10.29}$$

The Hamiltonian for general K can be expanded analogously, containing interactions of up to K spins.

The ground states of this Hamiltonian correspond to those assignments of all Boolean variables which violate the minimum number of clauses. In the satisfiable phase, the ground-state energy therefore equals zero, whereas it is strictly positive in the unsatisfiable phase. Consequently, if we were able to determine the ground-state energy as a function of the clauses-to-variables ratio α, we would also be able to identify the SAT/UNSAT threshold. In the statistical-mechanics approach, a weight $e^{-\beta\mathcal{H}}$ is assigned to every spin configuration, with $\beta = 1/T$ being a (formal) inverse temperature. For positive β this weight obviously favors assignments of low energy, and it gets more and more concentrated in such configurations if

$\beta \to \infty$. In the zero-temperature limit, i. e., for $\beta \to \infty$, only the ground states keep their non-zero probability of appearance. For our combinatorial problem of determining the satisfiability of a formula F leading to a specific Hamiltonian $\mathcal{H}$, we are therefore interested in the zero-temperature thermodynamical properties of the model.

In a series of pioneering papers [3–5], these ground-state properties were analyzed on the basis of a replica-symmetric approximation. The resulting threshold for 3-SAT, $\alpha_{\rm rs}(3) = 4.6$, is obviously larger than the numerical result, and replica-symmetry breaking effects had to be included. Including these into a variational approach [42], a second phase transition was found to exist inside the satisfiable phase, as is represented schematically in Fig. 10.9: Below some threshold $\alpha_d(K)$, the replica symmetry is exact. Descriptively this means that all solutions of a formula are collected in one large, connected cluster inside the configuration space $\{\pm 1\}^N$. At $\alpha_d(K)$, this cluster breaks discontinuously into an exponential number of smaller clusters, each one still containing an exponential number of solutions. Each pair of these clusters is separated by an extensive Hamming distance. The number of these clusters decreases with increasing α, until it eventually vanishes at the SAT/UNSAT transition $\alpha_c(K)$. Above this point, groundstates have non-zero energy, and almost all of them are collected in a sub-exponential number of clusters. The conjectured exact position of the two transitions for $K = 3$ were recently established using the cavity approach [6,7] at the level of one-step replica symmetry breaking. The clustering transition is located at $\alpha_d \simeq 3.92$, whereas the formulas become unsatisfiable with probability one at $\alpha_c \simeq 4.26$. For general K, the SAT/UNSAT threshold is given by [43]

$$\alpha_c(K) = \left(2^K - \frac{1}{2}\right) \ln 2 - \frac{1}{2} + \mathcal{O}(2^{-K}) \,, \tag{10.30}$$

i. e., for sufficiently large K it saturates the refined first-moment upper bound (10.21) up to some corrections which decrease exponentially in K. The clustering transition is located at

$$\alpha_d(K) = \frac{2^K}{K} \, \exp\left\{\frac{1}{2} e^{-\psi}\right\} \, \left(\ln K + \psi\right) \tag{10.31}$$

with ψ being the solution of

$$2 \, e^{\psi} = \psi + \ln K \,. \tag{10.32}$$

Compare this result with the asymptotic algorithmic lower bound (10.22). The dominating contributions coincide. It seems that local heuristic algorithms are not able to enter into the clustered phase. For an algorithm like simulated annealing or RandomWalkSAT this could be due to the proliferation of exponentially many local minima of $\mathcal{H}$ above $\alpha_d(K)$, for heuristic algorithms, which just assign one variable after the other – each variable at most once – this coincidence remains to be understood.

Within the cavity approach, it is also possible to go beyond the first step of replica-symmetry breaking [43,44]. The above picture has to be slightly refined. In between $\alpha_d(K)$ and $\alpha_c(K)$, a further threshold $\alpha_s(d)$ can be identified. For $\alpha \in (\alpha_d, \alpha_s)$, more than one step of RSB is present, and the solution clusters are themselves organized in clusters, etc. For $\alpha \in (\alpha_s, \alpha_c)$,

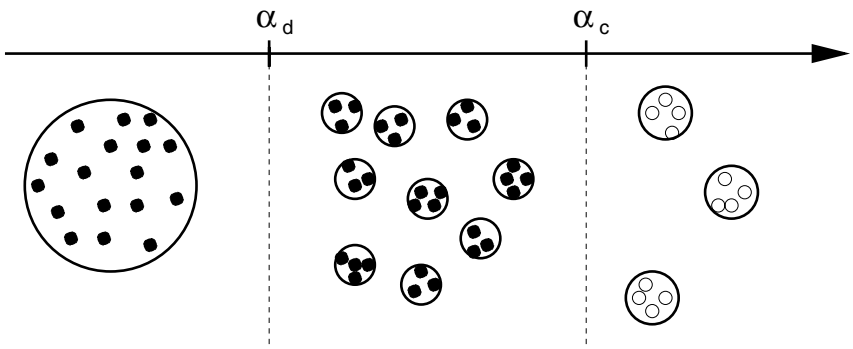


Figure 10.9: Schematic representation of the solution space structure of K-SAT. Below α_d, solutions are collected in one large cluster. Between α_d and α_c, an exponential number of solution clusters can be found. This clustering is accompanied by a proliferation of meta-stable states, i. e., local minima of $\mathcal{H}$. Above α_c, the formula is no longer satisfiable. Almost all ground states are collected in a sub-exponential number of clusters, and they have strictly positive energy.

this is no longer true. The picture is completely described by the first step of RSB. For $K = 3$ one finds $\alpha_s = 4.15$, asymptotically the ratio of α_s and α_d converges slowly towards one. For $K = 3$, the situation is still slightly more complicated, see [43, 44].

To summarize the results of this section, we have seen that methods borrowed from the statistical mechanics of disordered systems, like the replica and the cavity method, are frequently able to go beyond rigorous mathematical tools, giving detailed insight into the organization and the statistical properties of the satisfying assignments of a K-SAT formula. They allow us to establish conjectures for the exact location of phase transition thresholds, such as the SAT/UNSAT and the clustering threshold. As discussed in Sec. 7.4, many of these predictions are, however, still without any rigorous mathematical justification, which is an open challenge to mathematics.

10.3 Typical-case dynamics of RandomWalkSAT

In this section, we investigate the typical-case behavior of the RandomWalkSAT algorithm applied to random 3-SAT. The algorithm has already been presented in this chapter. All Boolean variables are assigned random truth values. Then, if existing, an unsatisfied clause is chosen randomly, and one again randomly selected variable in this clause is flipped. This step is repeated, until a solution is hit.

Of course, this algorithm works only for satisfiable formulas, for the case of random 3-SAT we have therefore to restrict our attention to the satisfiable phase, i. e., to clause-to-variable ratios $\alpha < 4.267$. If the formula is satisfiable, we have already seen that Schöning's analysis leads to a worst-case upper bound $T_{sol} = \mathcal{O}((4/3)^N)$ for the running time of RandomWalkSAT with restarts. This analysis was based on the assumption that there is at least one satisfying

assignment, and this specific assignment is hit by the algorithm. If we consider, however, random 3-SAT, there are exponentially many solutions. We therefore expect that a solution is *typically* hit after a much shorter time.

In the following, we are going first to give some numerical results, and then to present analytical approximations for the linear-time as well as the exponential-time behavior.

10.3.1 Numerical results

Numerically, we find a dynamical phase transition in the algorithmic behavior at about $\alpha_d \simeq 2.7$:

- For $\alpha < \alpha_d$, the algorithm almost surely finds a solution after *linear time*, we therefore call this region as the *easy SAT phase*. Consider Fig. 10.10. We have plotted the number M_u of unsatisfied clauses, divided by N, as a function of the rescaled time $t = T/N$, where T is the number of performed variable flips. Only for $\alpha < \alpha_d$, the curves reach zero in the observed timeinterval – and thus prove the satisfiability of the random 3-SAT formula.
- For $\alpha > \alpha_d$, the curves in Fig. 10.10 approach a non-zero plateau, i. e., the algorithm *equilibrates* to some $\alpha_u = M_u/N > 0$. Comparing the curves for different system sizes, cf. Fig. 10.11, we see that the value of α_u fluctuates around the plateau value, but extensive fluctuations are exponentially rare and are therefore strongly suppressed for larger formulas. These fluctuations go on until, after exponential time, a fluctuation is large enough to carry the system down to $\alpha_u = 0$, i. e., to a satisfying variable assignment. This exponential regime is called the *hard SAT phase*. *hard SAT phase*.

10.3.2 An analytical approximation scheme for random K-SAT

The final aim of an analytical approach should be the complete characterization of the probability $P(x_1, \ldots, x_N|T)$ in order to find the system after $T = tN$ algorithmic steps in the assignment $(x_1, \ldots, x_N)$. This task seems unfortunately far too complicated to be realized. In fact, the only rigorously known result on the behavior of RandomWalkSAT on random 3-SAT instances is that the algorithm is in fact running almost surely in linear time at least for $\alpha < 1.63$ [45]. This value therefore forms a lower bound on the dynamical transition α_d.

It is, however, possible to go beyond this result using a physical approximation method [46, 47]. The central idea is to characterize P by a finite set $\{\omega_1(x_1, \ldots, x_N), \ldots, \omega_n(x_1, \ldots, x_N)\}$ of global observables. We will explain below, which observables we use. We will assume the following properties.

- The observables become sharply concentrated in the limit $N \to \infty$, i. e., they follow almost surely the evolution of their averages. This assumption is analogous to one of the basic points for the applicability of Wormald's method in earlier chapters, and there should be no problem if the observables are densities, fractions, etc.

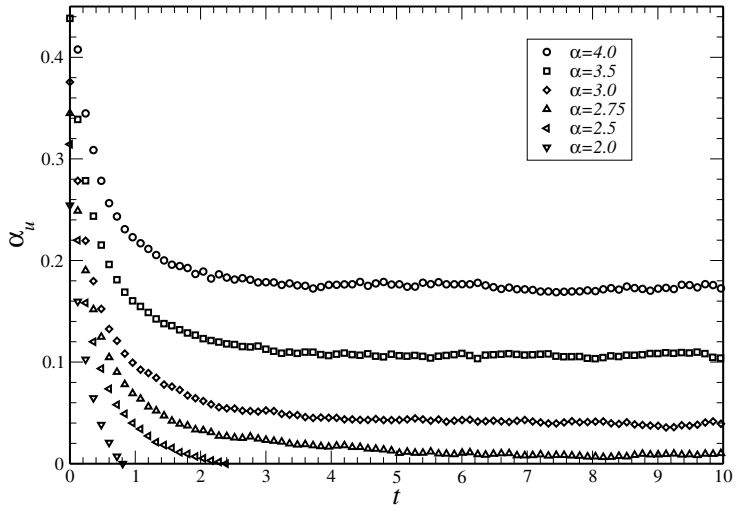


Figure 10.10: Energy density $\alpha_u = M_u/N$ as a function of the rescaled time t/N for $N = 50\,000$ and various values of α. For small enough α, a solution is found, whereas α_u equilibrates to a positive value for larger α.

- The distribution P depends on the assignment $(x_1, \ldots, x_N)$ only via $\{\omega_1(x_1, \ldots, x_N), \ldots, \omega_n(x_1, \ldots, x_N)\}$. This assumption is much more dangerous, and in fact it will hold, at most, approximately. The reason is that, in contrast to what we have seen in the case of the analysis of the linear-time algorithms for the q-core or for vertex covers, variables may be flipped more than once. The rigorous analysis of [45] is in fact based on those variables which can be flipped at most once (pure literals).

The simplest and most important observable in this context is the "energy" $\alpha_u(x_1, \ldots, x_N)$. It is very important because it allows us to detect satisfying assignments, and should therefore be contained in, or at least implied by, any reasonable set of observables. The simplest possible approximation, which will be developed in the following, is thus based only on this single observable.

Linear-time behavior

In Fig. 10.10, we have seen that the dynamical transition can be identified by looking to the evolution of the energy under a linear number of algorithmic steps, or as a function of a finite rescaled time. Below α_d, the energy reaches zero, above it equilibrates to a positive plateau value. It is therefore reasonable to start our investigation with finite t, and to investigate the

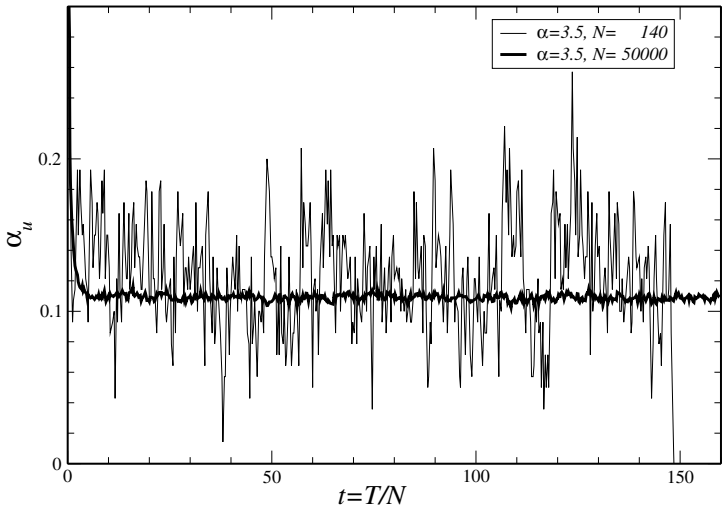


Figure 10.11: Energy density $\alpha_u = M_u/N$ as a function of the rescaled time t/N for $\alpha = 3.5$ and two different system sizes. It is obvious that both curves fluctuate around approximately the same plateau value, but fluctuations are suppressed for larger N.

evolution of the *average energy density*

$$\alpha_u(t) = \sum_{x_1,\ldots,x_N} \alpha_u(x_1,\ldots,x_N)\, P(x_1,\ldots,x_N|tN)\,. \tag{10.33}$$

If we flip one spin, three contributions to the energy have to be taken into account:

(i) The selected unsatisfied clause becomes satisfied, i. e., it gives a negative contribution -1 to the energy.

(ii) All other unsatisfied clauses containing the flipped variable become also satisfied. Within the approximation, there are, on average, $K\alpha_u(t)$ such clauses (note that a randomly selected variable is contained on average in $K\alpha$ clauses).

(iii) Each satisfied clause containing the flipped variable becomes unsatisfied if and only if it was satisfied only by this single variable. Within the approximation, the flipped variable is contained in $K(\alpha-\alpha_u(t))$ satisfied clauses, and a fraction $1/(2^K-1)$ of these clauses becomes unsatisfied.

Putting together these contributions, we find a closed approximate equation for $\alpha_u(t)$:

$$\begin{aligned}\frac{d}{dt}\alpha_u(t) &= \lim_{N\to\infty}\{M_u(T+1)-M_u(T)\} \\ &= -1 - K\alpha_u(t) + \frac{K(\alpha-\alpha_u(t))}{2^K-1}\,.\end{aligned} \tag{10.34}$$

The initial condition $\alpha_u(t=0) = 2^{-K}\alpha$ results from the fact that, in a random initial variable assignment, each clause in unsatisfied with probability 2^{-K} because all K contained variables have to be wrongly assigned. The solution of this simple linear ordinary differential equation is given by

$$\alpha_u(t) = \frac{\alpha}{2^K} - \frac{2^K - 1}{2^K K}\left[1 - \exp\left\{-\frac{2^K K}{2^K - 1}t\right\}\right] . \tag{10.35}$$

It describes an exponential decay from the initial condition to a final plateau value

$$\alpha_u(t \to \infty) = \frac{1 + \alpha K - 2^K}{2^K K} \tag{10.36}$$

with a relaxation time $\tau = (2^K - 1)/(2^K K)$. Note that this timescale does not depend on α, i. e., it is unchanged for formulas of different numbers of clauses, only the final energy density is α-dependent. This is, within the limits of numerical precision, consistent with Fig. 10.10.

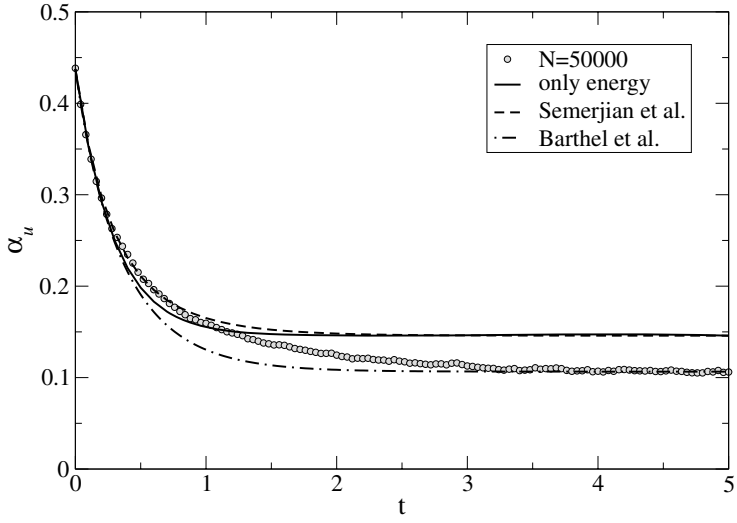


Figure 10.12: 3-SAT for $\alpha = 3.5$. Energy density $\alpha_u(t)$ for the various analytical approximations, together with a numerical result for $N = 50\,000$. The full line gives the energy approximation (10.35). The refined approximation of Semerjian and Monasson only improves the short-time behavior, whereas the refined approximation of Barthel *et al.* improves the plateau value.

The crucial result is that this equation shows a dynamical transition, when the plateau energy changes sign from negative, i. e., unreachable values for small α, to positive values for larger α. This transition is located at

$$\alpha_d = \frac{2^K - 1}{K} , \tag{10.37}$$

which, for $K = 3$, results in $7/3 = 2.\overline{3}$. This value is smaller than the numerical value 2.7, but given the crude energy approximation, it is still of satisfying quality. The reason for the failure can be seen in Fig. 10.12: Whereas the dynamics for very short times is well-represented, systematic deviations build up with ongoing algorithmic time. The analytical estimate of the plateau-value is too high, thus leading to a too-small value for the dynamical transition point. This difference seems to decrease for larger K, we therefore expect that Eq. (10.37) will become asymptotically exact for $K \gg 1$.

The qualitative behavior is, however, reproduced correctly. Below α_d, the energy curve reaches zero after a finite, α-dependent time t_{sol}, i. e., after a linear number $T_{sol} = t_{sol}N$ of algorithmic steps. This time diverges if we approach α_d from below. For larger values of α, the energy approaches a non-zero plateau, and no solution is found for a linear number of algorithmic steps.

The quality of the approximation can obviously be improved via the inclusion of a larger set $\{\omega_1(x_1, \ldots, x_N), \ldots, \omega_n(x_1, \ldots, x_N)\}$ of global observables. Two versions exist in the literature:

- Semerjian and Monasson [47] introduce the numbers of clauses being satisfied by $0, 1, \ldots, K$ variables, instead of distinguishing only between satisfied and unsatisfied ones. Their result is
$$\alpha_u(t) = \frac{\alpha}{2^K} - \frac{1}{K}\left[1 - \frac{1}{(1+\tanh t)^K}\right] , \tag{10.38}$$
as shown in Fig. 10.12 by the dashed line. This approximation obviously improves the short-time behavior, but the plateau height remains unchanged compared with the simpler energy approximation. This means also, that the estimate for the dynamical transition point does not vary.

- Barthel, Hartmann and Weigt [46] consider the fractions $p(s, u|t)$ of variables which appear in s satisfied and u unsatisfied clauses. Note that this quantity resembles the degree distributions studied in the case of heuristic VC algorithms. One consequently finds a set of differential equations, which has to be solved by numerical integration, see the result for $\alpha = 3.5$ in Fig. 10.12. We find that this solution nicely reproduces the asymptotic plateau energy, but the short-term behavior is not accurate, the approximation describes a faster decay than that found in direct simulations. In any case, the dynamical transition is given by the point where the plateau height vanishes, and it is given in full agreement with the numerical value at
$$\alpha_d(K = 3) = 2.71 . \tag{10.39}$$

A natural generalization of both approaches would consider the distribution $p(s_0, s_1, \ldots, s_K)$ of variables being contained in s_l clauses satisfied by l variables, with $l = 0, \ldots, K$. This ansatz can be expected to improve the short-term behavior as well as the plateau value, when compared with the simple energy approximation. Again the dynamical equations have to be integrated numerically, but the achieved agreement with the numerical short-term behavior is increased but is not yet perfect. In the case of $l \geq 1$, one could also take into account whether the clause is satisfied by the variable under consideration. Note, however, that all these improvements complicate the analysis, without ever being exact. The qualitative behavior, and parts of the quantitative behavior, are already met by simpler approximations.

Exponential-time behavior

For $\alpha > \alpha_d$, the solution time is exponential in N,

$$t_{sol}(\alpha) \propto e^{N\tau(\alpha)} \, . \tag{10.40}$$

We have, however, no clue as how to access these exponential timescales directly. The answer lies in considering *large deviations* from the typical behavior for finite times. More precisely, we are going to estimate the transition probability

$$P(\alpha_u(0) \to \alpha_u(t_f) = 0) \tag{10.41}$$

that the RandomWalkSAT, starting at $t = 0$ with initial energy density $\alpha_u(0)$, finds a solution after the *finite* time $t_f = \mathcal{O}(1)$, or, equivalently, after the linear number $t_f N$ of variable updates. How does this quantity help to infer $\tau(\alpha)$? Remember our numerical observation that solutions are found via fluctuations of the energy density around its plateau value. Large deviations, in particular one down to zero energy density, are exponentially rare. To actually find one such fluctuation, we have to almost surely wait for a time

$$t_{sol}(\alpha) \propto \frac{1}{P(\alpha_u(0) \to \alpha_u(t_f) = 0)} \tag{10.42}$$

with $1 \ll t_f \ll N$ (technically, the limit $N \to \infty$ is performed first, then t_f is also sent to infinity).

The transition probability (10.41) can be calculated following the lines shown in Sec. 8.3. Being at some time $t = T/N$ at energy level $\alpha_u(t)$, we have to calculate the probability of an energy change $\Delta e = M_u(T+1) - M_u(T)$:

$$P_T^{T+1}(\Delta e) = \sum_{s,u=0}^{\infty} e^{-K\alpha} \frac{[K(\alpha - \alpha_u(t))]^s [K\alpha_u(t)]^u}{s!\, u!} \sum_{l=0}^{s} \binom{s}{l} \mu^l (1-\mu)^{s-l} \delta_{\Delta e,\, l-u-1} \tag{10.43}$$

where μ denotes the probability that a satisfied clause was satisfied only by the flipped variable, and thus becomes unsatisfied. Within the energy approximation, we thus have to set $\mu = 1/(2^K - 1)$. The interpretation is the following. The energy change is composed of the three contributions *(i–iii)* discussed above in the context of the linear-time behavior. The flip for sure satisfies the selected, previously unsatisfied clause, i. e., $\Delta e_{(i)} = -1$. We have already discussed that, on average, there are $K\alpha_u(t)$ further unsatisfied clauses depending on the flipped variable. The energy approximation results in assuming the actual number u to be distributed according to a Poissonian, i. e., we have $\Delta e_{(ii)} = -u$ with probability $e^{-K\alpha_u(t)}[K\alpha_u(t)]^u/u!$. The last contribution $\Delta e_{(iii)}$ comes from those l under the, on average, $K(\alpha - \alpha_u(t))$ satisfied clauses which become unsatisfied under the spin flip. For a satisfied clause this happens with probability μ. All together, this results in a total energy change $\Delta e = \Delta e_{(i)} + \Delta e_{(ii)} + \Delta e_{(iii)}$ being (at least approximately) distributed according to Eq. (10.43).

Having written this quantity, we proceed in complete analogy to Sec. 8.3. Skipping the details of the calculation, we can express the transition probability as a path integral

$$P(\alpha_u(0) \to \alpha_u(t_f)) = \int_{\alpha_u(0)}^{\alpha_u(t_f)} \mathcal{D}\alpha_u(t) \int \mathcal{D}\kappa(t) \exp\left\{-N \int_0^{t_f} dt\, \mathcal{L}(\kappa(t), \alpha_u(t), \dot{\alpha}_u(t))\right\} \tag{10.44}$$

over all energy trajectories leading from $\alpha_u(0)$ to $\alpha_u(t_f)$, and over the conjugated parameter function $\kappa(t)$ which is not subject to initial or final conditions. The Lagrange function is given by

$$\mathcal{L} = -i\kappa(t)(1+\dot{\alpha}_u(t)) + K\alpha - K(\alpha - \alpha_u(t))\left[1-\mu(1-e^{-i\kappa(t)})\right] - K\alpha_u(t)e^{i\kappa(t)}\,. \tag{10.45}$$

For $N \to \infty$, the path integral in Eq. (10.44) is dominated by the "classical" trajectory, the latter being determined by the Euler–Lagrange equations

$$\begin{aligned} 0 &= \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\alpha}_u(t)} - \frac{\partial \mathcal{L}}{\partial \alpha_u(t)} \\ 0 &= \frac{\partial \mathcal{L}}{\partial \kappa(t)} \end{aligned} \tag{10.46}$$

which are given by

$$\begin{aligned} \dot{\alpha}_u(t) &= -1 - K\alpha_u(t)\, e^{i\kappa(t)} - K\mu\,(\alpha - \alpha_u(t))\, e^{-i\kappa(t)} \\ i\dot{\kappa}(t) &= K\, e^{i\kappa(t)} - K\,\left[1-\mu(1-e^{-i\kappa(t)})\right]\,. \end{aligned} \tag{10.47}$$

The solution

$$\begin{aligned} e^{i\kappa(t)} &= \frac{1+\mu A e^{2Kt}}{1-Ae^{2Kt}} \\ \alpha_u(t) &= \alpha_u(0)e^{-(1+\mu)Kt}\frac{1-Ae^{(1+\mu)Kt}}{1-A}\cdot\frac{1+\mu A e^{(1+\mu)Kt}}{1+\mu A} \\ &\quad + \int_0^t d\tau \left(-1+\mu K\alpha\frac{1-Ae^{(1+\mu)Kt}}{1+\mu Ae^{(1+\mu)Kt}}\right) e^{-(1+\mu)K(t-\tau)} \\ &\qquad \times \frac{1-Ae^{(1+\mu)Kt}}{1-Ae^{(1+\mu)K\tau}}\cdot\frac{1+\mu Ae^{(1+\mu)Kt}}{1+\mu Ae^{(1+\mu)K\tau}} \end{aligned} \tag{10.48}$$

already respects the initial condition $\alpha_u(0)$, and still depends on the supplementary parameter A which can be adjusted to fix the solution time t_f at which $\alpha_u(t)$ reaches zero.

The result is depicted for representative values of α and t_f in Fig. 10.13. Independently of the initial condition, the energy approaches exponentially fast the plateau (10.36) already calculated for the typical finite-time trajectory in the last subsection, with the same relaxation time $\tau = (2^K-1)/(2^K K)$ given there. It stays on this plateau until a short time before t_f. There it is constrained to deviate towards lower energies, until it finally reaches $\alpha_u(t_f) = 0$.

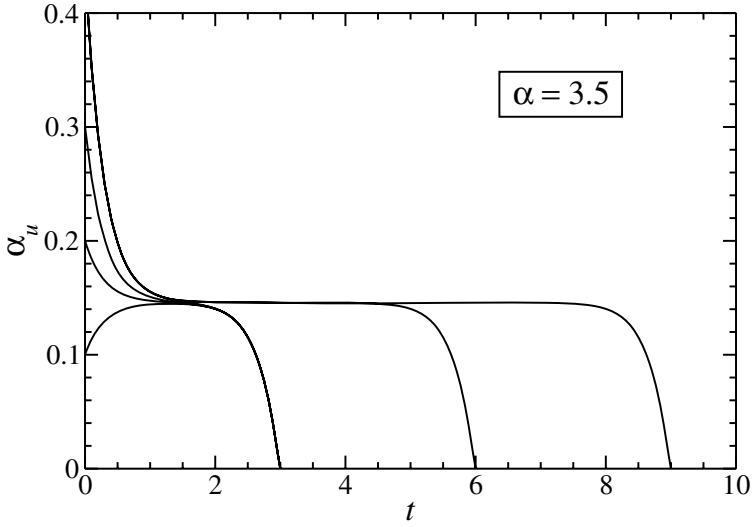


Figure 10.13: 3-SAT for $\alpha = 3.5$. Energy densities $\alpha_u(t)$ for various initial conditions $\alpha_u(0)$ and solution times t_f. The curves first approach a plateau value being almost independent of the initial condition, and deviate from this plateau only in a small final time-slice, to reach zero at t_f.

How does this result compare with the numerical one shown in Fig. 10.11? Consider, e. g., the curve for $N = 140$ which actually reaches a satisfying assignment in the graphically represented interval. This curve fluctuates wildly, frequently reaches energy values relatively close to zero, and also those which are more than twice as large as the plateau value. Nothing similar is to be seen in Fig. 10.13. The reason is that the latter curves have to be interpreted as an average trajectory over all trajectories solving the Boolean formula after $t_f N$ single-variable updates. All the fluctuations far before t_f are therefore averaged out, as fluctuations to higher and smaller energies are equally likely (this in fact determines the plateau value). Only the very last fluctuation which carries the system to a satisfying assignment of the Boolean variables is common to all finite-N trajectories – they are constrained to reach $\alpha_u = 0$ exactly at t_f. This fluctuation therefore does not average out, as can be seen in the figure. Therefore, the analytical result exactly represents the numerical findings.

To end this analysis, we can also estimate the exponential solution time within our energy approximation. According to Eqs (10.40), (10.42) and (10.44), it is given by the action of the classical trajectory,

$$\tau(\alpha) = \frac{1}{N} \ln t_{sol} = \lim_{t_f \to \infty} \int_0^{t_f} dt_f \; \mathcal{L}(\kappa(t), \alpha_u(t), \dot{\alpha}_u(t)) \; . \tag{10.49}$$

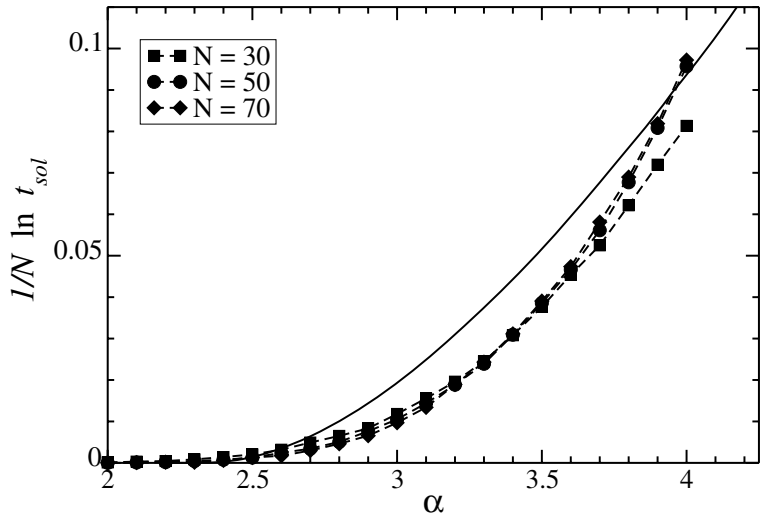


Figure 10.14: 3-SAT: Solution time t_{sol} of RandomWalkSAT as a function of α. The analytical result is given by the full line. Numerical data for $N = 30, 50, 70$ are represented by the symbols.

Results of a numerical integration are given in Fig. 10.14. We see that the coincidence with numerical data is not bad, given again, the crude energy approximation used. In addition, one has to consider that RandomWalkSAT is practically able to solve only very small problem instances. It is therefore not clear whether the numerical values, which are shown up to $N = 70$, already represent the asymptotic value well for large N.

To conclude Sec. 10.3, we have seen an approximation scheme for the typical-case dynamical behavior of the RandomWalkSAT algorithm. We have found a dynamical phase transition between an *easy SAT phase* for $\alpha < \alpha_d(K)$, where RandomWalkSAT almost surely solves the formula in a linear number of algorithmic steps, and a *hard SAT phase* for $\alpha > \alpha_d(K)$. In this phase, the algorithm first equilibrates to a positive fraction of unsatisfied clauses, and solves the formula only after an exponential waiting time due to a large, but rare fluctuation.

Note that, again, the onset of the hard-SAT phase depends on the algorithm. Using a certain fraction of greedy steps, it can be shifted to much larger values. It is conjectured, that the maximal value α_d reachable within SLS is related to the onset of non-trivial clustering, or more precisely to the proliferations of an exponential number of local energy minima. Since the algorithm does not satisfy physical conditions like detailed balance, this connection is, however, not clear. Establishing connections between the structure of the energy landscape, including number and organization of local and global minima, and the performance of an optimized local search, is one of the most challenging open questions in the field. It would represent a huge step forward in our understanding of the problem-intrinsic hardness of combinatorial optimization.

10.4 Message-passing algorithms for SAT

In this last section dedicated to the satisfiability problem, we will apply the ideas explained in Chap. 9 on message-passing algorithms to the K-SAT problem. As already mentioned, the survey propagation algorithm was in fact first proposed for solving K-SAT [6], and only later was also applied to other, more general problems.

There is one important difference between vertex cover, for which we have explained the fundamental ideas of message-passing, and K-SAT. VC consists of a large number of *a priori* easily satisfiable constraints. Each edge has to be covered, and this can be easily done by covering all vertices. The hardness of the problem results from the additional *global* constraint of covering a minimum number of vertices. The main combinatorial difficulty was therefore to extend minimum vertex covers of appropriate cavity graphs to the full graph, not all minimum vertex covers of the cavity graph were extensible to the full graph. For K-SAT, the situation is slightly more involved as the local constraints, i. e., the clauses, may be directly contradictory. As we will see below, the main task remains, however, to extend solutions for some cavity problem containing a reduced number of clauses, to the full problem instance under consideration.

10.4.1 Factor graphs and cavities

In order to use message-passing for a CNF formula, it is favorable to represent the formula by its *factor graph*:

Definition: *Factor graph*

Given a SAT formula F in conjunctive normal form, with N Boolean variables and M clauses, we define its *factor graph* G_F as the bipartite weighted graph $(V_{var} \cup V_{func}, E, c)$ with

- N variable nodes $i \in V_{var}$, each one corresponding to one Boolean variable,
- M function nodes $\mu \in V_{func}$, each one corresponding to one clause,
- edges $\{i, \mu\} \in E$ between variable and function nodes, if clause C_μ depends on variable x_i,
- edge weights $c_{\mu,i}$ defined in Eq. (10.26), identifying whether or not a variable appears negated in a clause.

An example is given below. Note that the neighbor set $N(i)$ of a variable node is now completely composed of function nodes, and vice versa.

Example: Factor and cavity graph for 3-SAT

The left-hand side of Fig. 10.15 shows the factor graph for the 3-SAT formula

$$F = (x_1 \vee \overline{x}_2 \vee x_4) \wedge (x_3 \vee \overline{x}_4 \vee x_5) \wedge (x_1 \vee \overline{x}_3 \vee x_6) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_6)$$

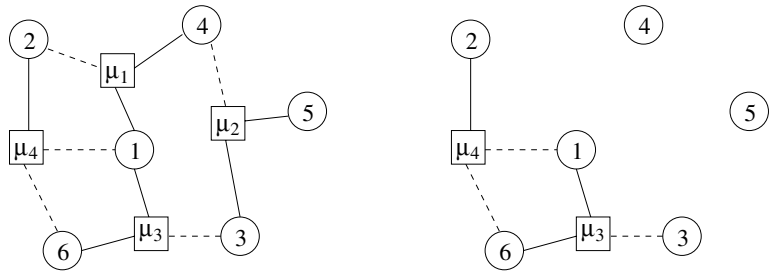


Figure 10.15: Factor graph G_F and cavity graph $G_F^{(4)}$ for the example. Variable nodes are represented by circles, function nodes by squares. If a variable i appears unnegated in a clause C_μ, we draw a full line for the edges $\{i, \mu\}$, if it appears negated, we draw a dashed line.

and contains six variable and four function nodes. For the cavity graph $G_F^{(4)}$, all clauses depending on x_4 (and $\overline{x}_4$) are removed. The cavity graph corresponds to the reduced formula

$$F^{(4)} = (x_1 \vee \overline{x}_3 \vee x_6) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_6)$$

which connects only two clauses. □

We will also need again the concept of a *cavity graph*:

Definition: *Cavity graph*

Given a factor graph G_F corresponding to formula F, and one variable node i, we define the *cavity graph* $G_F^{(i)} \subset G_F$ as the subgraph induced by the vertex set $V_{var} \cup V_{func} \setminus N(i)$.

Practically this means that we delete all function nodes μ which are adjacent to i, together with their incident edges. The cavity graph can also be considered as the factor graph of the *cavity formula*

$$F^{(i)} = \bigwedge_{\nu \in V_{func} \setminus N(i)} C_\nu \tag{10.50}$$

containing all clauses but those containing either x_i or its negation $\overline{x}_i$. In this formula i becomes isolated, and the variable x_i is not constrained by any clause. It can thus be set to any of the two values $\{0, 1\}$ without danger of violating the cavity formula.

Let us further denote the set of all satisfying assignments of the SAT formula F with $\mathcal{S}$, whereas $\mathcal{S}^{(i)}$ collects the solutions for the cavity formula $F^{(i)}$. Since every satisfying assignment of the full formula F satisfies $F^{(i)}$, too, we have $\mathcal{S} \subset \mathcal{S}^{(i)}$. The main issue of both message-passing algorithms which will be presented here consists in a scheme of how to see

which solutions out of $\mathcal{S}^{(i)}$ are also elements of $\mathcal{S}$, and how they can be characterized statistically. Again, an important classification of variables will be given by the fact that they are fixed to one specific truth value in all satisfying assignments (backbone), or if they change value from solution to solution (joker value).

10.4.2 Warning propagation

Let us consider an arbitrary variable assignment $\underline{x} = (x_1, \ldots, x_N)$. It does not satisfy formula F if there is at least one unsatisfied clause C_μ, for the latter we have $\sum_{i \in N(\mu)} c_{\mu,i}(2x_i - 1) = -K$ according to Eq. (10.27). This condition is fulfilled if and only if all variables x_i with $i \in N(\mu)$ fulfil $2x_i - 1 = -c_{\mu,i}$.

The variable assignment $\underline{x}$ is, on the other hand, satisfying if and only if for each clause C_μ at least one variable being assigned in the correct way, such that $2x_i - 1 = +c_{\mu,i}$ (or, equivalently, $x_i = (c_{\mu,i} + 1)/2$). We therefore define a warning from a function node to a variable node in the following way:

Definition: *Warning*

Given an edge $\{i, \mu\}$ of the factor graph G_F, the *warning* $u_{\mu \to i}(\underline{x})$ is defined as

$$u_{\mu \to i}(\underline{x}) = \prod_{j \in N(\mu) \setminus i} \delta_{-c_{\mu,i}, 2x_i - 1} \tag{10.51}$$

In simpler words, a warning sent from a function node μ to an adjacent variable node i equals zero, if the clause C_μ is already satisfied by at least one other variable $x_j,\ j \in N(\mu) \setminus i$, else the warning equals one. A non-zero warning thus contains the message to variable x_i: "The clause C_μ is not yet satisfied. Take value $(c_{\mu,i} + 1)/2$!"

For a set $\mathcal{M} \subset \{0, 1\}^N$ of Boolean assignments, the warning can be generalized to

$$u_{\mu \to i}(\mathcal{M}) = \min_{\underline{x} \in \mathcal{M}} u_{\mu \to i}(\underline{x}) \ . \tag{10.52}$$

Based on the warnings, we define a number of different local fields.

Definition: *Cavity field, local field, contradiction number*

- For any variable node i, the *local field* is given by

$$h_i = \sum_{\mu \in N(i)} c_{\mu,i}\, u_{\mu \to i} \ . \tag{10.53}$$

- For any variable node i, the *contradiction number* is given by

$$\chi_i = 1 - \delta_{|h_i|, \sum_{\mu \in N(i)} u_{\mu \to i}} \ . \tag{10.54}$$

- For any edge $\{i, \mu\} \in E$, the *cavity field* is the message from the variable to the function node given by

$$h_{i\to\mu} = \sum_{\nu\in N(i)\setminus\mu} c_{\nu,i}\, u_{\nu\to i}\ . \tag{10.55}$$

To understand the significance of these definitions, let us assume for a moment that we know one solution $\underline{x} \in \mathcal{S}^{(i)}$ of the cavity formula for an arbitrary, but fixed variable node i. We can thus determine the warnings $u_{\mu\to i}(\underline{x})$ for all its incident edges, and consequently also the local field h_i and the contradiction number χ_i. The interpretation of these quantities is now simple, they allow us to determine if $\underline{x}$ is a solution of the full formula F, if it can be modified to become such a solution by simply negating the assignment of x_i, or if it is not extensible to a solution of F.

- The field h_i sums all warnings sent to the variable node, weighted by the $c_{\mu,i}$. This means that, if h_i is negative, node i receives a majority of warnings forcing x_i to assume value zero, and if h_i is positive the majority of incoming warnings forces the variable to value one. For $h_i = 0$, the warnings are balanced, and both values of x_i are equally good. If x_i is set according to its local fields, a majority of the corresponding clauses C_μ is satisfied.

- Further on, the contradiction number is only zero, if there are no contradictory warnings forcing the variable to different values. Only in this case can variable x_i be assigned a value such that all warnings (and consequently the clauses C_μ connected to x_i) are satisfied.

To summarize this argument, $\underline{x}$ is not adaptable to a solution of F if the contradiction number χ_i is non-zero. If it is zero, the value of x_i has to be chosen in accordance with the local field h_i in order to find a satisfying assignment for the full SAT instance F.

The definition of the cavity field $h_{i\to\mu}$ is the same as for the local field h_i, except that the function node μ is ignored, i. e., "what is the local with of x_i without the presence of clause μ". The cavity field will be used below.

The problem is, of course, that for the moment we do not know any solution of the cavity problem $F^{(i)}$ of any i, determining one is computationally almost as hard as determining directly a satisfying assignment of F. The main aim of warning propagation is therefore to give a self-consistent scheme of determining all warnings $u_{\mu\to i}(\mathcal{S}^{(i)})$ for all edges $\{i, \mu\} \in E$.

Let us fix one edge $\{i, \mu\} \in E$, and consider the cavity graph $G_F^{(i)}$. Assume further on, that we know all warnings $u_{\nu\to j}(\mathcal{S}^{(j)})$ with $j \in N(\mu) \setminus i$ and $\nu \in N(j) \setminus \mu$, see Fig. 10.16 for an illustration of these neighborhood relations. These also define, following the previous definition, cavity fields $h_{j\to\mu}$ for all $j \in N(\mu) \setminus i$. We know that for these variable nodes in the cavity graph, we should set the x_j according to $\mathrm{sign}(h_{j\to\mu})$, in analogy with the case of the full graph discussed above. If we want to propagate this information from μ to i, we have to distinguish three cases:

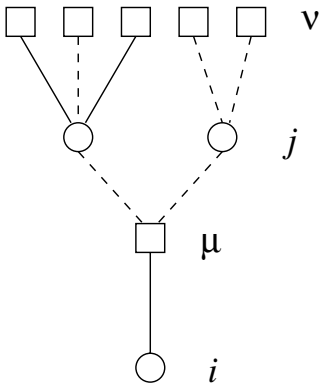


Figure 10.16: Illustration of the neighborhood relations to determine the warning $u_{\mu\to i}$ from the warnings $u_{\nu\to j}$.

- If there is $h_{j\to\mu} = 0$ for at least one $j \in N(\mu) \setminus i$, we can freely select the value of x_j. It can be set in particular to $(c_{\mu,j}+1)/2$, in this way satisfying clause C_μ. Consequently no non-trivial warning needs to be sent from μ to i.

- If all $h_{j\to\mu} \neq 0$, but at least one fulfils $\text{sign}(h_{j\to\mu}) = c_{\mu,i}$, the cavity formula $F^{(j)}$ already forces the corresponding variable x_j to a value automatically satisfying C_μ. Consequently no non-trivial warning needs to be sent from μ to i.

- If none of the two previous cases is true, all variables x_j with $j \in N(\mu) \setminus i$ are forced by their cavity graphs to take values which do not satisfy C_μ. The function node has to sent $u_{\mu\to i} = 1$ in order to force x_i to satisfy C_μ.

Summarizing these cases, we can write the resulting warning as

$$u_{\mu\to i}(\mathcal{S}^{(i)}) = \prod_{j\in N(\mu)\setminus i} \delta_{-c_{\mu,j},\text{sign}(h_{j\to\mu})} \,. \tag{10.56}$$

Note that using simultaneously the $u_{\nu\to j}(\mathcal{S}^{(j)})$ for various j is correct only if we assume statistical independence of these variable nodes j on the cavity graph $G_F^{(i)}$.

This argument can be applied to all $\mu \in N(i)$, and thus results in a closed set of equations for the warnings $u_{\mu\to i}$ and the cavity fields $h_{i\to\mu}$. If we are able to solve these, we finally can calculate all h_i and χ_i in order to determine if F is also satisfiable, and which truth value has to be assigned to which variable.

We can summarize the above findings to the *warning propagation* equations (WP):

$$\begin{aligned} h_{j\to\mu} &= \sum_{\nu\in N(j)\setminus\mu} c_{\nu,j}\, u_{\nu\to j} \\ u_{\mu\to i} &= \prod_{j\in N(\mu)\setminus i} \delta_{-c_{\mu,j},\mathrm{sign}(h_{j\to\mu})} \\ h_i &= \sum_{\mu\in N(i)} c_{\mu,i}\, u_{\mu\to i} \\ \chi_i &= 1-\delta_{|h_i|,\sum_{\mu\in N(i)} u_{\mu\to i}}\,. \end{aligned} \tag{10.57}$$

The algorithmic utility of these equations is completely analogous to the one for vertex cover, cf. Chap. 9. First the warnings and cavity fields are determined iteratively, than the local fields and the contradiction numbers are calculated. If, on one hand, the contradiction number is one for at least one variable, WP predicts unsatisfiability for formula F. If, on the other hand, all contradiction numbers vanish, WP predicts satisfiability, and variables can be assigned values according to the local fields in an iterative decimation procedure.

The problems of WP are the same for vertex cover and SAT. The iterative update of the equations for warnings and cavity fields may fail to converge for two reasons. The first is short loops which are violating the independence approximations of nodes in the cavity graph. The second reason is the possibility of exponentially many solutions of the WP equations. In this case the algorithm locally tries to converge to different solutions in different regions of the factor graph, and global convergence is prevented. For this latter case, survey propagation is useful.

10.4.3 Survey propagation

As discussed in Chap. 9, the existence of many solutions of the WP equations indicates that the solution set $\mathcal{S}$ is clustered into a large number $\mathcal{N}_{cl}$ of macroscopically separated clusters $\mathcal{S}_\ell$, $\ell = 1,\ldots,\mathcal{N}_{cl}$. According to the discussion of Sec. 10.2, this is almost surely the case in a non-zero interval $\alpha \in (\alpha_d(K), \alpha_c(K))$. In this region, we have to use the survey propagation algorithm once more.

As in Chap. 9, we define warnings and fields cluster-wise, and consider histograms. In particular we introduce, for each edge in the factor graph, the *survey*

$$Q_{\mu\to i}(u) = \frac{1}{\mathcal{N}_{cl}^{(i)}} \sum_{\ell=1}^{\mathcal{N}_{cl}^{(i)}} \delta_{u,u_{\mu\to i}(\mathcal{S}_\ell^{(i)})} \tag{10.58}$$

where we have summed over all solution clusters of the cavity formula $F^{(i)}$. Now we want to construct the resulting distribution of cavity fields $h_{i\to\mu}$ resulting from the warnings $u_{\nu\to i}$ with $\nu \in N(i)\setminus\mu$ according to Eq. (10.55). To do so, we need the joint distribution of

all these warnings. As usual, we assume maximal independence between these warnings, hence the joint probability of the $\{u_{\nu\to i}\}$ contains a product over all $\nu \in N(i) \setminus \mu$ of the $Q_{\nu\to i}(u_{\nu\to i})$. We are, however, talking about *satisfying* assignments only, so the set of the possible combinations $\{u_{\nu\to i}\}$ is restricted to be non-contradictory: $u_{\nu_1\to i}u_{\nu_2\to i} = 0$ if $c_{\nu_1,i} \neq c_{\nu_2,i}$ for all $\nu_1, \nu_2 \in N(i) \setminus \mu$. At this point, it is useful to introduce the sets $N_+(i) = \{\mu \in N(i) \mid c_{\mu,i} = +1\}$ ($N_-(i) = \{\mu \in N(i) \mid c_{\mu,i} = -1\}$) as the set of all function nodes for clauses containing variable x_i directly (negated). Using this notation, we can write the joint probability as

$$\frac{1}{Z_{i\to\mu}}\left[\prod_{\nu\in N(i)\setminus\mu} Q_{\nu\to i}(u_{\nu\to i})\right] \quad \Big[\ \Theta\Big(\sum_{\nu\in N_+(i)\setminus\mu} u_{\nu\to i}\Big)\prod_{\nu'\in N_-(i)\setminus\mu}\delta_{0,u_{\nu'\to i}} + \Theta\Big(\sum_{\nu\in N_-(i)\setminus\mu} u_{\nu\to i}\Big)\prod_{\nu'\in N_+(i)\setminus\mu}\delta_{0,u_{\nu'\to i}} + \prod_{\nu\in N(i)\setminus\mu}\delta_{0,u_{\nu\to i}}\ \Big] \tag{10.59}$$

which contains, in the second bracket in each line, one of the contributions leading to positive, negative and zero cavity fields, picking out only the non-contradictory combinations of warnings. Θ is the step function which takes value one if the argument is strictly positive, and zero otherwise. The normalization constant $Z_{i\to\mu}$ can be written as

$$Z_{i\to\mu} = (\Pi^{+1}_{i\to\mu} + \Pi^{-1}_{i\to\mu} + \Pi^{*}_{i\to\mu}) \tag{10.60}$$

with

$$\begin{aligned}
\Pi^{+1}_{i\to\mu} &= \sum_{\{u_{\nu\to i}\}}\left[\prod_{\nu\in N(i)\setminus\mu} Q_{\nu\to i}(u_{\nu\to i})\right]\Theta\Big(\sum_{\nu\in N_+(i)\setminus\mu} u_{\nu\to i}\Big)\prod_{\nu'\in N_-(i)\setminus\mu}\delta_{0,u_{\nu'\to i}}\\
\Pi^{-1}_{i\to\mu} &= \sum_{\{u_{\nu\to i}\}}\left[\prod_{\nu\in N(i)\setminus\mu} Q_{\nu\to i}(u_{\nu\to i})\right]\Theta\Big(\sum_{\nu\in N_-(i)\setminus\mu} u_{\nu\to i}\Big)\prod_{\nu'\in N_+(i)\setminus\mu}\delta_{0,u_{\nu'\to i}}\\
\Pi^{*}_{i\to\mu} &= \sum_{\{u_{\nu\to i}\}}\left[\prod_{\nu\in N(i)\setminus\mu} Q_{\nu\to i}(u_{\nu\to i})\right]\prod_{\nu\in N(i)\setminus\mu}\delta_{0,u_{\nu\to i}}\,.
\end{aligned} \tag{10.61}$$

Note that the u can only take values in $\{0, 1\}$, i. e., we may write

$$Q_{\mu\to i}(u) = (1 - \eta_{\mu\to i})\delta_{u,0} + \eta_{\mu\to i}\delta_{u,1} \tag{10.62}$$

by introducing the probability $\eta_{\mu\to i}$ that a non-trivial message is sent. Doing so, the last

expressions simplify,

$$\begin{aligned}
\Pi^{+1}_{i\to\mu} &= \left[1-\prod_{\nu\in N_+(i)\setminus\mu}(1-\eta_{\nu\to i})\right]\prod_{\nu'\in N_-(i)\setminus\mu}(1-\eta_{\nu'\to i})\\
\Pi^{-1}_{i\to\mu} &= \left[1-\prod_{\nu\in N_-(i)\setminus\mu}(1-\eta_{\nu\to i})\right]\prod_{\nu'\in N_+(i)\setminus\mu}(1-\eta_{\nu'\to i})\\
\Pi^{*}_{i\to\mu} &= \prod_{\nu\in N(i)\setminus\mu}(1-\eta_{\nu\to i})\,. \qquad (10.63)
\end{aligned}$$

From these quantities we can read off that, e. g., the probability of having a cavity field $h_{i\to\mu}\geq 1$ equals $\Pi^{+1}_{i\to\mu}/(\Pi^{+1}_{i\to\mu}+\Pi^{+1}_{i\to\mu}+\Pi^{*}_{i\to\mu})$, and analogous relations are valid for negative and vanishing cavity fields.

How can we propagate these messages? The idea is similar to that of warning propagation. Only if *all* cavity fields $h_{j\to\mu}$ with $j\in N(\mu)\setminus i$ forces the spins x_j to assume truth values not satisfying clause C_μ, i. e., only if $\mathrm{sign}(h_{j\to\mu})=-c_{\mu,j}$ for all these j, is a non-trivial warning sent from $\mu\to i$, and x_i is forced to take the value satisfying C_μ. As this happens by definition with probability $\eta_{\mu\to i}$ for a randomly selected solution cluster, we deduce

$$\eta_{\mu\to i}=\prod_{j\in N(\mu)\setminus i}\left[\frac{\Pi^{-c_{\mu,j}}_{j\to\mu}}{\Pi^{+1}_{j\to\mu}+\Pi^{+1}_{j\to\mu}+\Pi^{*}_{j\to\mu}}\right]. \qquad (10.64)$$

Equations (10.63) and (10.64) form the first part of the *survey-propagation* (SP) equations which have to be solved iteratively in the sense explained in Chap. 9.

All the quantities discussed so far in this section concern cavity fields and warnings. The statistical properties of the full solution set $\mathcal{S}$ of a formula F are, however, described by the effective local fields. From Chap. 9 we already know that these can be calculated by analogy with the cavity fields, the only difference being that warnings from all neighbors are to be taken into account. This results in the generalization of Eqs (10.63) to

$$\begin{aligned}
\Pi^{+1}_{i} &= \left[1-\prod_{\mu\in N_+(i)}(1-\eta_{\mu\to i})\right]\prod_{\mu'\in N_-(i)}(1-\eta_{\mu'\to i})\\
\Pi^{-1}_{i} &= \left[1-\prod_{\mu\in N_-(i)}(1-\eta_{\mu\to i})\right]\prod_{\mu'\in N_+(i)}(1-\eta_{\mu'\to i})\\
\Pi^{*}_{i} &= \prod_{\mu\in N(i)}(1-\eta_{\mu\to i})\,. \qquad (10.65)
\end{aligned}$$

These quantities allow us to calculate the *total biases*

$$\begin{aligned} W_i^{+1} &= \frac{\Pi_i^{+1}}{\Pi_i^{+1}+\Pi_i^{-1}+\Pi_i^*} \\ W_i^{-1} &= \frac{\Pi_i^{-1}}{\Pi_i^{+1}+\Pi_i^{-1}+\Pi_i^*} \\ W_i^{*} &= \frac{\Pi_i^{*}}{\Pi_i^{+1}+\Pi_i^{-1}+\Pi_i^*} \end{aligned} \tag{10.66}$$

describing the probability that vertex i has a positive / negative / vanishing field in a randomly selected cluster of solutions of the full SAT formula F. It thus also equals the probability that the corresponding Boolean variable x_i is fixed to one / fixed to zero / unfixed, within the assignments collected in this cluster. These biases can be calculated once we have converged the iteration of the warning probabilities $\eta_{\mu\to i}$, and they can be exploited algorithmically to construct a satisfying assignment. Note that in contrast to warning propagation, we do not need to calculate contradiction numbers, since contradictions are explicitly eliminated in the above equations.

The simplest reasonable heuristic to select and fix variables in this context is a greedy one. Select a vertex i of maximal total bias W_i^{+1} or W_i^{-1}, i. e., a variable which is frozen to the same truth value in a maximum number of clusters. Set x_i to this truth value, and reduce the formula via UCP, see the beginning of Sec. 10.1. For the smaller formula, the warnings can be reiterated according to the SP equations, and the decimation process can be iterated. Note that this procedure can be extremely accelerated by fixing not only one variable, but a certain small percentage of the most biased variables. This decreases the computational complexity of the algorithm from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \ln N)$ (resulting from the need to sort the biases), but it unfortunately also decreases the success rate of the decimation procedure.

There are three possible outcomes of this decimation procedure, run on a satisfiable formula:

1. The algorithm fixes variables until all clauses are satisfied. The not-yet-fixed variables may be assigned arbitrary values, and a solution is found.
2. Survey propagation converges at a certain stage to the trivial solution $\eta_{\mu\to i} \equiv 0$ for all $\{\mu, i\} \in E$, which corresponds to an unclustered and unfrozen set of satisfying assignments of the decimated SAT formula. In this case, SP is of no more use, but the reduced problem can be passed to a state-of-the-art SLS algorithm (like modern versions of WalkSAT), and is found to be easily solvable in most cases.
3. The SP algorithm does not converge, even if the initial problem was satisfiable. This happens possibly only after some decimation steps.

In practice, the second of these three cases is observed most frequently when run inside the hard SAT region $(\alpha_d(K), \alpha_c(K))$. In this case, huge formulas ($N \leq 10^7$) can be satisfied. If we approach, however, the SAT/UNSAT threshold, the third and unsuccessful case starts to dominate. Close to the transition, where formulas are critically constrained, the errors made due to the approximate character of SP (independence assumption on the cavity graph), connected to the simple nature of the greedy decimation heuristic, start to play a crucial

role. In this region, better strategies including, e. g., backtracking, should be developed to make SP applicable. To give an example, SP works efficiently for 3-SAT in an interval for α which equals for large enough random formulas $(3.92, 4.25)$, only in the very small interval up to the SAT/UNSAT threshold at 4.267 does it fail to output a satisfying assignment.

We have started this chapter by explaining some complete as well as some stochastic algorithms for K-SAT and by analyzing their worst-case running times. Then we have discussed some recent progress in the understanding of the typical-case behavior of random K-SAT, which is the most prominent problem in complexity theory in general, and in the development of a typical-case theory in particular. We have seen that statistical-mechanics methods allow us to make fascinating predictions about the character and location of phase transitions in the solvability of combinatorial problems and in the solution-time behavior of algorithms. These insights finally allowed the development of a new type of statistical-mechanics motivated algorithm. Still, many important points in this context remain open. First, existing statistical-mechanics results still frequently do not have a mathematically rigorous confirmation. Second, there remain open questions even within understanding of the statistical mechanics, e. g., we do not know the solution-space structure beyond the first step of replica-symmetry breaking, or the connection between this structure and the failure of local polynomial-time algorithms. Third, algorithms like survey propagation are currently applicable only to problems having a locally more or less tree-like structure, whereas realistic problems are frequently characterized by the existence of loops on various length scales. These and many other questions are still to be answered.

Bibliography

[1] D. Mitchell, B. Selman, and H. Levesque, in: *Proc. of the 10th Natl. Conf. on Artificial Intelligence AAAI-92*, 459 (1992).

[2] B. Selman and S. Kirkpatrick, Science (Washington DC) **264**, 1297 (1994).

[3] R. Monasson and R. Zecchina, Phys. Rev. Lett. **76**, 3881 (1996).

[4] R. Monasson and R. Zecchina, Phys. Rev. E **56**, 1357 (1997).

[5] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky, Nature **400**, 133 (1999).

[6] M. Mézard, G. Parisi, and R. Zecchina, Science **297**, 812 (2002).

[7] M. Mézard and R. Zecchina, Phys. Rev. E **66**, 056126 (2002).

[8] M. Mézard and G. Parisi, Eur. Phys. J. B **20**, 217 (2001).

[9] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah, in: D. Du, J. Gu, and P. Pardalos (eds), DIMACS Series in Discrete Mathematics and Theoretical Computer Science **35**, 19 (American Mathematical Society, Providence 1997).

[10] B. Aspvall, M. F. Plass, and R. E. Tarjan, Inf. Proc. Lett. **8**, 121 (1979).

[11] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, Reading (MA) 1974).

[12] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms – Theory and Practice*, (Prentice-Hall, Englewood Cliffs (NJ) 1977).

[13] M. Davis and H. Putnam, J. Assoc. Comp. Machin. **7**, 201 (1960).

[14] M. Davis, G. Logemann, and D. Loveland, Commun. ACM **5**, 394 (1962).

[15] I. Lynce and J. P. Marques-Silva, Ann. Math. Artif. Intell. **37**, 307 (2003).

[16] R. M. Stallman and G. J. Sussman, Artif. Intell. **9**, 135 (1977).

[17] R. Dechter, Artif. Intell. **41**, 273 (1989/90).

[18] B. Monien and E. Speckenmeyer, Discr. Appl. Math. **10**, 287 (1985).

[19] R. Rodosek, in: J. Calmet, J. A. Campbell, and J. Pfalzgraf (eds.), *Proc. 3rd Intern. Conf. on AI and Symbolic Math. Computation*, Lecture Notes on Computer Science **1138**, 197 (Springer, Heidelberg 1996).

[20] E. Dantsin, A. Goerdt, E. A. Hirsch, and U. Schöning, in: U. Montanari *et al.* (eds.), *Proc. Int. Colloq. on Automata, Languages and Programming ICALP'00*, Lecture Notes in Computer Science **1853**, 236 (Springer, Heidelberg 2000).

[21] E. Dantsin *et al.*, Theor. Comp. Sci. **289**, 69 (2002).

[22] T. Stützle, H. Hoos, and Andrea Roli, Technical Report AIDA-01-02, Darmstadt University of Technology, Computer Science Department, Intellectics Group (2001).

[23] C. H. Papadimitriou, in: *Proc. 32nd Ann. IEEE Symp. on Found. of Comp. Sci. FOCS'91*, 163 (1991).

[24] L. E. Reichl, *A Modern Course in Statistical Physics*, (John Wiley & Sons, New York 1998).

[25] B. Selman, H. Kautz, and B. Cohen, in: D. S. Johnson and M. A. Trick (eds.), *Cliques, Coloring and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science vol. **26**, 337 (American Mathematical Society, Providence 1996).

[26] U. Schöning, in: *Proc. 40th Ann. IEEE Symp. on Found. of Comp. Sci. FOCS'99*, 410 (IEEE, New York 1999).

[27] U. Schöning, Algorithmica **32**, 615 (2002).

[28] W. Feller, *An Inroduction to Probability Theory and its Applications*, (Wiley, New York 1968).

[29] G. R. Grimmet and D. R. Stirzaker, *Probability and Random Processes*, (Oxford University Press, Oxford 1992).

[30] R. Schuler, U. Schöning, and O. Watanabe, Res. Rep. Math. Comp. Sci. Ser. C **146**, 1 (2001).

[31] T. Hofmeister, U. Schöning, R. Schuler, and O. Watanabe, in: H. Alt and A. Ferreira (eds.), *Proc. 19th Ann. Symp. Theoretical Aspects Computer Science STACS 2002*, 192 (Spinger, Berlin 2002).

[32] S. Baumer and R. Schuler, in: E. Giunchiglia and A. Tacchella (eds.), *Theory and Applications of Satisfiability Testing. 6th Intl. Conf. SAT 2003*, 150 (Springer, Berlin 2004).

[33] E. Friedgut, J. Amer. Math. Soc. **12**, 1017 (1999).

[34] O. Dubois and Y. Boufkhad, J. Algorithms **24**, 395 (1997).

[35] L. M. Kirousis, E. Kranakis, and D. Krizanc, Rand. Struct. Alg. **12**, 253 (1998).

[36] Y. Boufkhad, O. Dubois, and J. Mandler, Electr. Coll. Comput. Compl. **10**, 07 (2003).

[37] M. T. Chao and J. Franco, SIAM Journal on Computing **15**, 1106 (1986); Information Science **51**, 289 (1990).

[38] D. Achlioptas, Theor. Comp. Sci. **265**, 159 (2001).

[39] J. Franco, Theor. Comp. Sci. **265**, 147 (2001).

[40] A. C. Kaporis, L. M. Kirousis, and E. G. Lalas, in: R. H. Möhring and R. Ramam (eds.), *Algorithms – ESA 2002, Proceedings"* (Springer, New York 2002).

[41] D. Achlioptas and Y. Peres, Journal of the AMS **17**, 947 (2004).

[42] G. Biroli, M. Weigt, and R. Monasson, Eur. Phys. J. B **14**, 551 (2000).

[43] S. Mertens, M. Mézard, and R. Zecchina, arXiv preprint `cs.CC/0309020`.

[44] A. Montanari, G. Parisi, and F. Ricci-Tersenghi, J. Phys. A **37**, 2073 (2004).

[45] Alekhnovich, Ben-Sasson, proc. FOCS'02, 352 (2003); Electr. Coll. Comput. Compl. **11**, 16 (2004).

[46] W. Barthel, A. K. Hartmann, and M. Weigt, Phys. Rev. E **67**, 066104 (2003).

[47] G. Semerjian and R. Monasson, Phys. Rev. E **67**, 066103 (2003).

11 Optimization problems in physics

In the previous chapters, we have seen that one can learn a great deal about combinatorial optimization problems when applying concepts and methods from statistical physics. This means physics is useful for theoretical and practical computer science. This is not totally new, because a couple of different heuristic optimization methods were invented in the field of statistical physics. The first and most famous one is the *simulated annealing* approach invented by Kirkpatrick, Gelatt and Vecchi [1]. Such heuristic approaches are very useful in the case where no fast, i. e., polynomial algorithms are available. These heuristics, although most of the cases give only approximations of the true optima, have the advantage that they usually can be applied to various types of problem. Some of the more recent methods are outlined in the first part of this chapter, details can be found in Refs [2, 3].

The transfer of knowledge does not proceed only in one direction, from physics to computer science. On the contrary, many high-level tools from computer science are applied in physics more and more frequently. This is true in particular for combinatorial optimization algorithms. During the last two decades there has been much progress in the development of highly efficient algorithms, which led to an strong growth in the application of these techniques in physics. In the second part of this chapter, we give a short overview over the latest developments in this field [2–5].

The most natural application for a combinatorial optimization algorithm is the calculation of the ground state (GS) of a system, i. e., the behavior at absolute zero temperature, $T = 0$. This is especially useful for systems exhibiting a *quenched disorder*, e. g., systems described by some random parameters such as interaction constants or local fields. The term "quenched" means that these random parameters are chosen at the beginning for each realization of the disorder and then they are kept fixed during a calculation or simulation. The behavior of each realization, also for the GS, depends crucially on the quenched disorder, hence one usually has to study many realizations to investigate the behavior of a model. This is similar to the ensembles of random graphs or random satisfiability problems which we have studied before. Below, we will discuss e. g., spin glasses, random-field systems, proteins, and polymers in random media.

Since exact zero temperature is not reachable in experiments, one could ask the question, why one should be interested in GS calculations. In statistical physics and several other fields, obtaining ground states is indeed quite useful. We give a number of reasons for this:

- If the ground states are obtained by an exact algorithm, then one is sure that it represents the true thermodynamic equilibrium behavior at $T = 0$. Therefore, one has no problems with equilibration in comparison with using, e. g., Monte Carlo simulations.

- Analytical theories, especially for disordered system, usually involve some approximations. The results of the theories can very often be extrapolated to zero temperature. Hence, obtaining true ground states allows us to verify the approximations which have been made.
- If the system under investigation exhibits a phase transition, one can very often extract useful information about the critical exponents from ground-state calculations.

  The most obvious case is when the phase transition happens at zero temperature, as for two-dimensional Ising spin glasses. In this case, one performs the calculations right at the phase transition.

  In some cases of random systems, the phase transitions occurring at finite temperatures (with moderate disorder) are governed by the zero-temperature transitions driven by stronger disorder. This can be seen in renormalization-group (RG) calculations of these systems. One example is the *random-field Ising ferromagnet* where, for a Gaussian disorder, the behavior along the full transition line is governed by the zero-temperature fixed point of the RG calculations. This system will be considered in Sec. 11.5.
- Sometimes one might expect that the behavior found at zero temperature persists at finite, but low, temperatures. One example of this are geometrical properties of domains found in disordered Ising magnets.
- Ground-state calculations can also be useful to extract information about excited states, i. e., true $T > 0$ behavior. In this case one first calculates a ground state. Then one modifies the realization of the disorder in such a way that the ground state is energetically more disfavored than excited states. Hence, the ground state of the modified system will be an excited state of the original system, which can therefore be obtained by an additional ground-state calculation of the modified system. As an example we will consider below droplet and domain-wall excitations in *two-dimensional spin glasses*.
- There are also cases, where even more information can be obtained about a physical system by using optimization algorithms. Below we will present one example, where the full partition function of the $q \to \infty$ *Potts model* can be calculated by optimization of *submodular functions*.
- Sometimes the optimization is just used as a tool to analyze and understand a physical system, e. g., when a model is matched to given data. The classical example is to fit functions to a set of parameters. A more complex example is the estimation of parameters for interacting galaxies, as referenced in the section on *genetic algorithms*, below.

In this chapter, we first introduce three heuristic optimization methods which can be applied to almost all types of optimization problems. In the first section, more advanced *Monte Carlo methods* are treated, in the second section, *hysteric optimization* and in the third, *genetic algorithms*. The first two methods have originated in physics, while the third one has a biological background; we include it here, because it has been used for many applications in physics. The second part of this chapter is dedicated to applications in physics of polynomial-time optimization algorithms developed in computer science, namely *shortest-path algorithms*, *maximum-flow algorithms*, optimization of *submodular functions* and *matching algorithms*.

The emphasis will be on the physical background and on the mapping of the physical systems to the corresponding optimization problems. Usually, we will only outline the basic idea of the optimization algorithms and mention a few important physical results. For more details on the algorithms, the reader should consult the mentioned computer science literature or Refs [2, 3], while for more results concerning the physical systems, we refer the reader to the cited research papers.

11.1 Monte Carlo optimization

A general introduction to Monte Carlo (MC) methods has already been given in Sec. 6.6 in the context of the vertex-cover problem. For general introduction to MC simulations see, e. g., the textbooks by Newmann/Barkema [6] and by Landau/Binder [7]. In this section some additional techniques relevant for optimization problems are explained. First, we explain the *simulated annealing* method, which is a general method to obtain low-energetic states using MC simulations. Next, *cluster algorithms* are introduced, which allow us to speed up MC simulations significantly. Finally, *biased sampling* techniques are explained, which are methods used to tailor the simulations towards specific targets.

11.1.1 Simulated annealing

The basic idea of simulated annealing [1, 8, 9] goes back to an experimental technique, for growing a crystal with as few defects as possible. This works by slowly cooling down the probes such that defects and other lattice impurities (forming meta-stable states) can heal out and a pure crystalline structure (the global minimum) is achieved at low temperature. In this process the temperature, which represents kinetic energy allowing jumps over energy barriers between various configurations, has to decrease very slowly since otherwise the molecular configuration gets stuck and the crystal will be imperfect. This is called annealing, in materials science. We have already presented a related approach in Sec. 6.6, that of compactification, for obtaining high-density configurations of the hard-core gas.

For simulated annealing, one maps the optimization problem under consideration on a physical system, identifies the cost function with the Hamiltonian $\mathcal{H}$ of the system and the variables $\mathbf{x}$ of the cost function with the physical degrees of freedom. Also, corresponding to elementary changes $\mathbf{y} \to \mathbf{z}$ of the variables, one chooses some dynamics in the physical system. Then the system is simulated using standard Monte Carlo methods, see Sec. 6.6, at a given temperature T, the statistical weights given in this case by the Boltzmann factors

$$P(\mathbf{x}) = \frac{1}{Z} \exp(-\mathcal{H}(\mathbf{x})/T)\,, \tag{11.1}$$

where Z is the canonical partition function $Z = \sum_{\{\mathbf{x}\}} \exp(-\mathcal{H}(\mathbf{x})/T)$. Within simulated annealing, initially the temperature is chosen at a high value, then T is gradually lowered, with the hope of finding ground states for $T \to 0$.

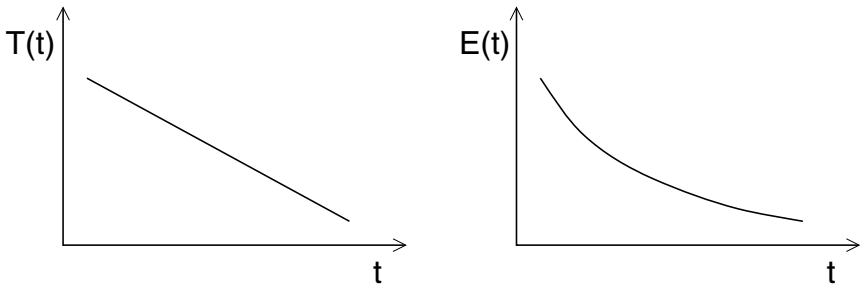


Figure 11.1: Simulated annealing. A linear cooling schedule $T(t)$ (left) and a sketch of the resulting average energy $E(t)$ as a function of time typically.

Obviously it will not be possible to equilibrate arbitrarily large systems at arbitrarily low temperatures, with only a finite amount of computer time available. The crucial ingredient for a good performance of the simulated annealing procedure will therefore be the cooling protocol $T(t)$, where T is the temperature and t is a measure of the computer time (e. g., the number of MC sweeps through the system). With the help of the theory of Markov processes it has been shown that, in principle, cooling protocols do exist with which simulated annealing finds the optimal solution of a given (finite) problem in infinite time. These cooling protocols have the form

$$T(t) = \frac{a}{b + \log(t)} \tag{11.2}$$

where a and b are positive constants that depend on the special problem. Practical applications, of course, have to be finished in finite time and therefore prefer the following empirical cooling protocols. The *linear* scheme (Fig. 11.1, left)

$$T(t) = a - bt \tag{11.3}$$

where a is the starting temperature and b the step size for decreasing the temperature (usually $0.01 \leq b \leq 0.2$); and the *exponential* scheme:

$$T(t) = ab^t \tag{11.4}$$

where again a is the starting temperature and b the cooling rate (usually $0.8 \leq b \leq 0.999$). On the right of Fig. 11.1, a typical development of the energy with time, during a cooling experiment, is shown.

Usually, due to the finiteness of the simulation time, simulated annealing allows us to obtain configurations close to the ground states, but not true ground states. This is sufficient for most applications. If one is interested in better results, while exact ground states are not necessary, one can switch to genetic algorithms, see Sec. 11.3, which usually are superior to simulated annealing, although they require more programming and a good choice of additional moves in configuration space. Another way of improving the performance, while still applying MC

simulations, is to use *parallel tempering* [10, 11], i. e., to simulate independent copies of the system at several temperatures in "parallel", while occasionally exchanging the copies between different temperatures. Since each individual copy is subjected to varying temperatures, sometimes the temperature is high, allowing high-energy barriers to be overcome and thus to escape local minima. Details of simulated tempering have already been given in Sec. 6.6.4 for the case of the hard-core gas, i. e., for the grand-canonical ensemble. The scheme translates in a simple way to the canonical ensemble, as can be easily worked out by the reader, or see Ref. [2].

11.1.2 Cluster algorithms

The most simple MC algorithms usually used for simulated annealing or parallel tempering are based on moves consisting of "atomic" changes of the configurations, e. g., insertion and removal of particles for the hard-core gas described in Sec. 6.6. These moves are accepted with certain probabilities, leading to transition rates fulfilling detailed balance, Eq. (6.12), resulting in a sampling of the desired distribution $P(.)$. Under several circumstances, these "atomic" moves have a too-small effect. This happens, e. g., when simulating close to phase transitions, where the correlation length diverges, or when using MC simulations as optimization techniques in connection with simulated annealing, as discussed in the previous subsection. *Cluster algorithms* involve larger-scale changes of the configurations, allowing for faster equilibration. These algorithms have been used, for example, for classical spin models, such as the Ising model [12], for lattice quantum systems [13] or for systems of hard particles [14]. Here, we will introduce cluster algorithms using the example of a simple Ising ferromagnet, i. e., Ising spins $\sigma_i = \pm 1$ interacting with their nearest neighbors, described by the Hamiltonian

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j \quad J > 0. \tag{11.5}$$

The system is simulated in the canonical ensemble, hence the probability of occurrence for a configuration $\mathbf{x} = \{\sigma_i\}$ at temperature T is given by Eq. (11.1). An MC move consists usually of two parts. Given a current configuration $\mathbf{y}$, first a "neighboring" trial configuration $\mathbf{z}$ is generated, usually randomly, and then $\mathbf{z}$ is accepted with a certain probability resulting in the move $\mathbf{y} \to \mathbf{z}$. If $\mathbf{z}$ is not accepted, then the configuration $\mathbf{y}$ is kept, i. e., the move $\mathbf{y} \to \mathbf{y}$ is performed. To understand the cluster approach better, it is useful to rewrite the transition probabilities in such a way, that this two-step process is reflected [15]:

$$W(\mathbf{y} \to \mathbf{z}) = A(\mathbf{y} \to \mathbf{z}) \tilde{W}(\mathbf{y} \to \mathbf{z}) \quad (\mathbf{y} \neq \mathbf{z}) \,. \tag{11.6}$$

$A(\mathbf{y} \to \mathbf{z})$ is the probability that $\mathbf{z}$ is the generated trial configuration and $\tilde{W}(\mathbf{y} \to \mathbf{z})$ the probability that the trial configuration is accepted. The probability that the current configuration $\mathbf{y}$ is kept, is still given by $W(\mathbf{y} \to \mathbf{y}) = 1 - \sum_{\mathbf{z} \neq \mathbf{y}} W(\mathbf{y} \to \mathbf{z})$. Inserting Eq. (11.6) into the detailed balance condition Eq. (6.12) yields, after a slight rearrangement,

$$\frac{\tilde{W}(\mathbf{y} \to \mathbf{z})}{\tilde{W}(\mathbf{z} \to \mathbf{y})} = \frac{P(\mathbf{z})}{P(\mathbf{y})} \frac{A(\mathbf{z} \to \mathbf{y})}{A(\mathbf{y} \to \mathbf{z})} \,. \tag{11.7}$$

One way to ensure detailed balance is to use the Metropolis algorithm [16], i. e., choosing

$$\tilde{W}(\mathbf{y} \to \mathbf{z}) = \min\left(1, \frac{P(\mathbf{z})}{P(\mathbf{y})} \frac{A(\mathbf{z} \to \mathbf{y})}{A(\mathbf{y} \to \mathbf{z})}\right) , \tag{11.8}$$

The reader may easily convince himself that detailed balance (11.7) is fulfilled.

For a single spin-flip dynamics in an N-spin system, usually one selects a spin randomly, which is flipped in the trial configuration, i. e., $A(\mathbf{y} \to \mathbf{z}) = 1/N$ for configurations differing by an orientation of exactly one spin, and $A(\mathbf{y} \to \mathbf{z}) = 0$ else. This yields for the acceptance probability (Metropolis) for these neighboring configurations

$$\tilde{W}(\mathbf{y} \to \mathbf{z}) = \min\left(1, \frac{P(\mathbf{z})}{P(\mathbf{y})}\right) = \min\left(1, \exp(-[\mathcal{H}(\mathbf{z}) - \mathcal{H}(\mathbf{y})]/T)\right) . \tag{11.9}$$

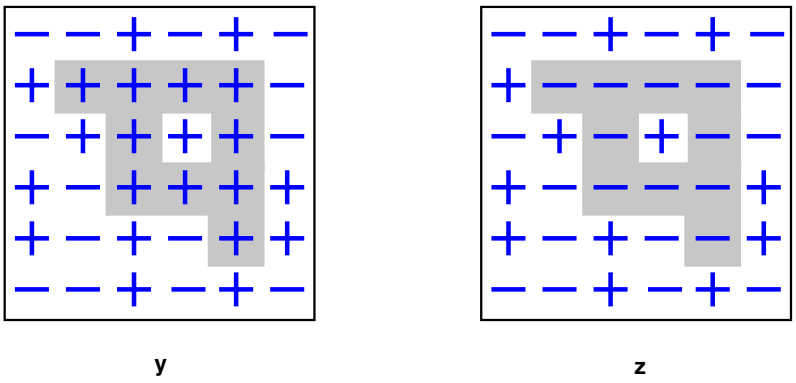


Figure 11.2: Example for the Wolff cluster algorithm. The shaded area denotes the cluster of spins, which is flipped when moving between configurations $\mathbf{y}$ and $\mathbf{z}$.

The basic idea of a cluster algorithm is that a larger fraction of spins, i. e., a cluster, is flipped in one step. In the example shown in Fig. 11.2, the cluster which is flipped when moving between configurations $\mathbf{y}$ and $\mathbf{z}$ is indicated by the shaded area. Here we explain the Wolff algorithm [12]. Within this algorithm, a cluster is started by selecting one spin i_0 randomly, all spins have the same probability $1/N$ of being selected. Then neighboring spins of the same orientation as σ_{i_0} are added to the cluster iteratively. This is done by *activating* bonds $\{i, j\}$ connecting a spin i, which is already part of the cluster, to neighbor spins j not (yet) part of the cluster. A bond is activated with probability p if $\sigma_j = \sigma_i$, and it is never activated if $\sigma_j \neq \sigma_i$. Below we will determine a suitable value for p, leading to a high efficiency. The final cluster consist of all spins connected by activated bonds. Each bond touching the cluster has exactly *one* "chance" to be activated. Note that a spin, which has the same orientation as the cluster spins and which is adjacent to several cluster spins, has *several* "chances" to be added to the cluster, because it is sufficient that one of the bonds to the cluster spins is activated. Hence, if the spin is adjacent to k spins of the final cluster, it will not be part of the cluster, only with probability $(1 - p)^k$. The + spin inside the cluster in configuration $\mathbf{y}$ in Fig. 11.2 has $k = 4$.

When the cluster construction has stopped, we denote with $n_{\rm same}$ the number of non-activated bonds to neighbors of the cluster, i. e., across the cluster boundary, which have the same orientation as the clusters spins, but which are not part of the cluster. Note that these bonds contribute negatively to the total energy in configuration $\mathbf{y}$, i. e., these bonds are *satisfied*. For configuration $\mathbf{y}$ in Fig. 11.2, the cluster of + spins has $n_{\rm same} = 13$ non-activated bonds to non-cluster neighbors with + orientation. Furthermore, for later use, we denote by $n_{\rm diff}$ the number of bonds from the cluster to neighbors having a different orientation from the cluster spins, i. e., the unsatisfied bonds across the cluster boundary ($n_{\rm diff} = 7$ in configuration $\mathbf{y}$ in the example).

Since the cluster construction stops when all adjacent satisfied bonds have not been activated, this happens with probability $(1-p)^{n_{\rm same}}$. Hence, the probability that a cluster in configuration $\mathbf{y}$ has been built, i. e., the probability that configuration $\mathbf{z}$ is chosen as a trial configuration, can be written in the form

$$A(\mathbf{y} \to \mathbf{z}) = A_{\rm interior} \times (1-p)^{n_{\rm same}} \tag{11.10}$$

where $A_{\rm interior}$ is the probability that the spins, which are part of the cluster, have been added to the cluster. This probability is a complicated sum over all different possibilities, which lead to the cluster. The good news is that we do not need to know $A_{\rm interior}$ to find an efficient algorithm. Also, we see that $n_{\rm diff}$ does not appear explicitly in $A(\mathbf{y} \to \mathbf{z})$.

For the reverse move $\mathbf{z} \to \mathbf{y}$, the non-activated bonds, which connect to spins having the orientation of the clusters spins, and the bonds to spins having a different orientation, exchange their identities compared with the move $\mathbf{y}$ to $\mathbf{z}$. Hence, the number of non-activated satisfied bonds across the cluster boundary for configuration $\mathbf{z}$ is exactly $n_{\rm diff}$ (the number of unsatisfied bonds across the cluster boundary of configuration $\mathbf{y}$). This results in the following probability that the cluster is constructed in configuration $\mathbf{z}$

$$A(\mathbf{z} \to \mathbf{y}) = A_{\rm interior} \times (1-p)^{n_{\rm same}}\,, \tag{11.11}$$

with $A_{\rm interior}$ being exactly the same as for the cluster construction in configuration $\mathbf{y}$, because the relative orientation of the cluster spins to each other, remains the same in both configurations $\mathbf{y}$ and $\mathbf{z}$.

To evaluate Eq. (11.8), we have to calculate the energies of configurations $\mathbf{y}$ and $\mathbf{z}$. We decompose the energy into three contributions, $E_{\rm interior}$ between the members of the cluster, $E_{\rm exterior}$ between the spins not part of the cluster, and the energy arising from the interactions between cluster and non-cluster spins, i. e., from the bonds across the cluster boundary. Note that $E_{\rm interior}$ and $E_{\rm exterior}$ are the same for $\mathbf{y}$ and $\mathbf{z}$, because the relative orientation of the spins inside the cluster is the same, and the spins outside the cluster do not change their orientations. As already noted, by flipping the cluster, the satisfied and unsatisfied bonds across the cluster boundary exchange their roles. Therefore, we get for the energies:

$$\mathcal{H}(\mathbf{y}) = E_{\rm interior} + E_{\rm exterior} - n_{\rm same} J + n_{\rm diff} J \tag{11.12}$$

$$\mathcal{H}(\mathbf{z}) = E_{\rm interior} + E_{\rm exterior} + n_{\rm same} J - n_{\rm diff} J\,. \tag{11.13}$$

Inserting Eqs (11.10), (11.11), (11.12), and (11.13) into Eq. (11.8) yields for the acceptance probability of the move $\mathbf{y} \to \mathbf{z}$, with $E_{\rm interior}$, $E_{\rm exterior}$ and $A_{\rm interior}$ conveniently dropping

out:

$$\begin{aligned}\tilde{W}(\mathbf{y}\to\mathbf{z}) &= \min\left\{1, \frac{e^{n_{\mathrm{diff}}J/T}e^{-n_{\mathrm{same}}J/T}}{e^{-n_{\mathrm{diff}}J/T}e^{n_{\mathrm{same}}J/T}}\frac{(1-p)^{n_{\mathrm{diff}}}}{(1-p)^{n_{\mathrm{same}}}}\right\}\\ &= \min\left\{1, \left[\frac{e^{-2J/T}}{1-p}\right]^{n_{\mathrm{same}}}\left[\frac{1-p}{e^{-2J/T}}\right]^{n_{\mathrm{diff}}}\right\}. \end{aligned} \tag{11.14}$$

If we choose $p = 1 - e^{-2/T}$ we obtain $\tilde{W}(y \to z) = 1$, i. e., *all* moves will be accepted. This means the Wolff cluster algorithm is rejection-free. It allows for a fast simulation close to the phase transition point, where the clusters percolate. For high temperatures $T \to \infty$, it follows that $p \to 0$, hence the clusters will be smaller, ultimately reducing to single-spin flip dynamics again. For $T \to 0$, we obtain $p \to 1$. Since at low temperatures the system will be ordered, almost all spins have the same orientation, i. e., the cluster will comprise almost all spins. This means again that only few spins are changed relative to the large cluster, hence effective single-spin flip dynamics is obtained in this limit as well. Nevertheless, when applying simulated annealing, together with the cluster algorithm, and starting at a temperature well above the transition temperature, one will very quickly find the ferromagnetic ground state, opposite to when using simulated annealing with single-spin flip dynamics.

Since the ferromagnet is not of particular interest within the frame of optimization problems, several extensions of cluster algorithms for other models have been tried. For the RFIM (see Sec. 11.5) a cluster algorithm [17] has been developed, but it is much less efficient than the Wolff algorithm for the ferromagnetic Ising model. Since there are polynomial-time GS algorithms for the RFIM, the cluster algorithm is not very useful in this case. Even worse is the situation for the NP-complete problem of finding GSs in spin glasses. It is possible to formulate the Wolff algorithm for this case, by activating satisfied bonds when building the cluster. Unfortunately, the algorithm turns out to be inefficient [18, 19], because the clusters start to percolate at a very high temperature, much above the spin-glass transition temperature. Hence, at low temperatures, one basically flips all spins. Thus, for frustrated spin systems, there is still some work to do to find a good cluster MC algorithm.

For a different type of system, consisting of hard particles, e. g., hard disks, cluster algorithms have been very useful [20]. The cluster methods work differently for those systems, because the particle interactions are not described by an energy function, similar to the hard-core systems discussed in the context of the vertex-cover problem. Here again, the grand-canonical ensemble provides a suitable background for describing these systems. The basic idea of the cluster algorithm [14] is to generate a copy of the configuration, which is rotated by 180° around some randomly chosen pivot point. Then the system and the rotated copy are "merged" and clusters of overlapping particles are built. The positions of the particles forming connected components can be exchanged between the original configuration and the rotated configuration. This move corresponds to a rotation of a cluster of particles of the system, where the cluster is chosen by construction such that it can be rotated *without* leading to overlaps of particles. The construction guarantees that overlapping particles are just taken into the rotated cluster, which actually avoids the overlap. This works very well for very high particle densities in glassy systems. Only when approaching maximum densities, does the same effect as for the Wolff algorithm occur. The clusters start to fill out the whole system, i. e., the full

system is rotated, which does not lead to a physically different configuration. More details can be found in Ref. [15].

11.1.3 Biased sampling

Within MC simulations, one usually wants to measure averages of observable quantities $A(\mathbf{y})$

$$\langle A\rangle := \sum_{\mathbf{y}} A(\mathbf{y})P(\mathbf{y})\,, \tag{11.15}$$

e. g., the average magnetization of the ferromagnetic Ising model discussed before. As we have seen in Sec. 6.6.2, importance sampling consists of generating sample configurations $\{\mathbf{y}^i\}$ according to the distribution $P(.)$ and then taking an arithmetic mean. This can be done, e. g., by using the MC approach. Suppose now, that we have a way to generate sample configurations according a *different* probability distribution $Q(.)$. Supposing $Q(\mathbf{x}) > 0\ \forall \mathbf{x}$, we can rewrite Eq. (11.15) as [21]

$$\langle A\rangle = \sum_{\mathbf{y}} A(\mathbf{y})P(\mathbf{y}) = \sum_{\mathbf{y}} A(\mathbf{y})\frac{P(\mathbf{y})}{Q(\mathbf{y})}Q(\mathbf{y}) = \langle AP/Q\rangle_Q\,, \tag{11.16}$$

where $\langle\ldots\rangle_Q$ denotes the average according to the probability Q. This means, to calculate the average of A with respect to $P(.)$, we can also generate configurations $\{\mathbf{z}^i\}$ according to the probabilities $Q(.)$ and then take an average of the quantities $\{A(\mathbf{z}^i)P(\mathbf{z}^i)/Q(\mathbf{z}^i)\}$. Hence, by introducing the distribution $Q(.)$ one imposes a *bias* on the configurations towards the configurations favored by $Q(.)$.

A direct application of biased sampling is *variance reduction*. The variance

$$\sigma^2(A) := \sum_{\mathbf{y}} (A(\mathbf{y}) - \langle A\rangle)^2 P(\mathbf{y}) = \langle A^2\rangle - \langle A\rangle^2 \tag{11.17}$$

of the measured quantity A is of great interest, because it is related to a simple error estimate $\sigma(A)/\sqrt{(M-1)}$ of $\langle A\rangle$, where M is the number of independent sample configurations $\{\mathbf{y}^i\}$. If we are able to reduce the variance $\sigma^2(A)$, we will have a more accurate estimate of the mean $\langle A\rangle$. The variance under probability distribution $Q(.)$ of AP/Q is given by

$$\begin{aligned}\sigma_Q^2(AP/Q) &:= \sum_{\mathbf{y}} \left(\frac{A(\mathbf{y})P(\mathbf{y})}{Q(\mathbf{y})} - \left\langle\frac{AP}{Q}\right\rangle_Q\right)^2 Q(\mathbf{y}) \\ &= \sum_{\mathbf{y}} \left(\frac{A(\mathbf{y})P(\mathbf{y})}{Q(\mathbf{y})} - \langle A\rangle\right)^2 Q(\mathbf{y})\,.\end{aligned} \tag{11.18}$$

Suppose now that we already knew the true answer $\langle A\rangle$. Then we would be able to choose

$$Q(\mathbf{x}) = \frac{A(\mathbf{x})P(\mathbf{x})}{\langle A\rangle}\,. \tag{11.19}$$

Inserting this into Eq. (11.18) results in $\sigma_Q^2(AP/Q) = 0$, hence the measurement will be arbitrarily accurate. This is not surprising, because the exact result is already known and plugged into the sampling in such a way that, in Eq. (11.16), always just one value, the answer $\langle A \rangle$, is sampled. This results trivially in obtaining as the mean, the exact answer with zero variance. Although this example is just academic, Eq. (11.19) provides a guideline for the case where the exact answer is not known. One should sample in the region where the measurable quantity A is large. This is also a kind of importance sampling, tailored to the specific quantity one is interested in. Nevertheless, designing an algorithm which performs this sampling according Eq. (11.19) is also often a significant challenge.

Introducing a bias also helps, when one performs MC simulations to obtain low-energy or ground states. Usual MC simulations often get stuck at low temperatures in local minima. As we have seen for parallel tempering, sometimes visiting high-energy states helps to overcome the barriers surrounding the local minima. Another approach to achieve this, is to perform a micro-canonical sampling, where the histogram obtained in energy space is flat, i. e., all energies contribute with the same weight. The idea behind this approach is that the simulation performs a random walk in energy space, hence higher-energy configurations are visited quite frequently. A particularly simple algorithm implementing this idea has been proposed recently by Wang and Landau [22]. Similar methods of generating a biased or even flat histogram in energy space are the *umbrella sampling* technique [23] and the *multi-canonical* approach [24].

The Wang–Landau method works by generating configurations having energy $E = \mathcal{H}(\mathbf{x})$ with a weight proportional to $1/g(E)$, where $g(E)$ is the density of states, i. e., $P(\mathbf{x}) \sim 1/g(\mathcal{H}(\mathbf{x}))$. This is done by accepting moves $\mathbf{y} \to \mathbf{z}$ with the Metropolis acceptance probability

$$\tilde{W}(\mathbf{y} \to \mathbf{z}) = \min\left\{1, \frac{g(\mathcal{H}(\mathbf{y}))}{g(\mathcal{H}(\mathbf{z}))}\right\} . \tag{11.20}$$

The simulation is performed within a desired range $[E_{\min}, E_{\max}]$ of energies. Since the density of states is not known in advance, one initially sets $g(E) = 1$ for all $E \in [E_{\min}, E_{\max}]$. One starts the simulation with this density of states. Also a histogram $K(E)$ is stored, counting how often each energy level is visited. Each time an energy E is visited, $K(E) \leftarrow K(E) + 1$ and the density of states is updated according to a multiplicative scheme $g(E) \leftarrow g(E)f$, with f some pre-chosen factor. Initially f is large, e. g., $f = \exp(1)$. The simulation is run until the histogram $K(E)$ of energies is flat, i. e., each energy level has been encountered (within some fluctuations) the same number of times. Then the factor f is reduced ($f \leftarrow \sqrt{f}$), the histogram of energies is reset to zero, and the simulation continues. This is repeated until the factor f is close to one, e. g., $f = \exp(10^{-8})$ (corresponding roughly to reducing f 27 times). Note that the resulting density of states represents only the relative density. The absolute numbers can be obtained by matching to a known result (e. g., $g(E_0) = 2$ at the ground-state energy level E_0 of an Ising ferromagnet).

For unfrustrated models like the ferromagnetic Ising model, the Wang–Landau approach has been proven to be quite useful when, e. g., obtaining the partition function by integrating over the density of states [22]. Also, ground states are visited frequently within the simulations. Unfortunately, for frustrated systems like spin glasses, the approach seems to have problems [25] at low temperatures in comparison with heuristic algorithms like genetic algorithms. The

reason is that, although energy barriers can be overcome easily, the algorithm gets stuck in entropic traps, usually present in frustrated systems. Furthermore, for larger systems, one has to partition the desired energy range into subranges, which are independently simulated. Hence, each copy is *not* able to visit the full energy range to overcome energy barriers. Here a combination with a parallel tempering-type approach might help. Nevertheless, since the method is quite new, it might still turn out to be useful for hard optimization problems, in particular when combined with other methods.

Another approach, which is based on biased sampling, is the *pruned-enriched Rosenbluth method* (PERM) [26]. This method is used to generate ground states (i. e., *native states*) of proteins within a simple lattice model, the so-called HP model [27, 28]. A protein in this model is a linear self-avoiding chain of hydrophobic (H) and polar (P) particles placed on the sites of a simple-cubic lattice. The energy function counts the number of neighboring H particles, each contributing an energy $-\epsilon$. The native state consists of maximizing the number of H$-$H contacts, leading to configurations, where the H particles are collected together in the center of the folded chain. This mimics the behavior of proteins in a water solvent, where hydrophobic amino acids prefer to have no contact with the solvent.

To generate chains of the HP model, *chain-growth* algorithms are used. This means one places a particle on a randomly chosen site and builds the chain, according to the given sequence of H and P particles, by adding one particle after the other at the current end of the chain. For simple sampling, to obey the self-avoidance constraint, one has to throw away a chain intersecting itself. This algorithm leads to a probability for successful generation of a chain, which is exponentially small in the length N of the chain. The biased growth of the chain grows only in directions where the self-avoidance is not violated.[1] Additionally, using a technique called *enrichment*, the energetic contributions arising from the H$-$H contacts are taken into account by another bias. This works by considering not only one chain, but a collection of chains, where chains carrying originally a higher weight occur more often in the sample than others, but with reduced weights. A pedagogical presentation, including the details, can be found in Ref. [2].

Using PERM at low temperatures, low-energetic configurations for HP proteins of length of the order $N = 100$ particles were found [29, 30], which have a lower energy than the results obtained previously by other standard MC methods. This again shows that biased sampling is sometimes useful in the framework of optimization problems.

11.2 Hysteric optimization

When putting a ferromagnet into an external magnetic field at low enough temperature, the system will align many spins with the field. If the system had no net magnetization before, the presence of the field will increase its magnetization. If, on the other hand, the system had a net magnetization opposite to the external field, the absolute value of the magnetization will first decrease for a while. If one waits long enough, the ferromagnet will flip more and more spins

[1] Although sometimes it cannot be avoided for a chain to get stuck and the sample then has to be disregarded.

in the direction of the field, hence again increasing the magnetization after a while. In general, the response of a ferromagnet to a field depends on the history of the system, the temperature, the material properties and on also the time.

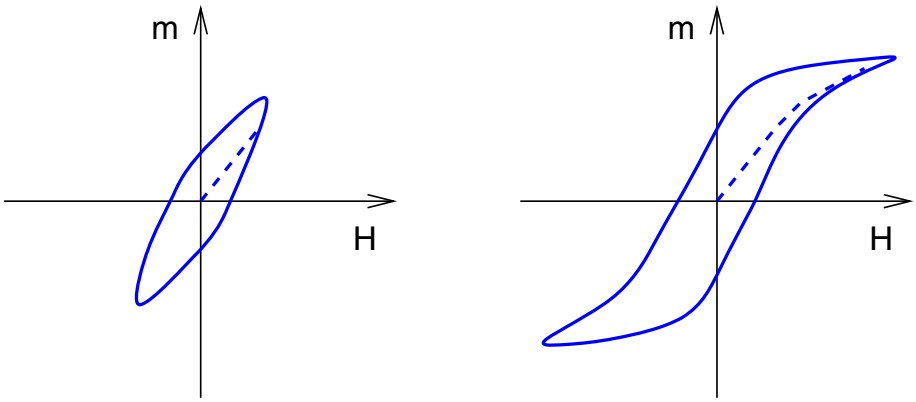


Figure 11.3: Hysteresis curves for ferromagnets, with a small (left) and large (right) amplitude $\hat{H}$ of the external magnetic field. The dashed lines show the behavior when starting with an initially demagnetized system.

A systematic way to probe the time- and history-dependent reaction to an external magnetic field is to perform a *hysteresis*. This means one subjects the system to a magnetic field $H(t)$ which varies with time t, e. g., $H(t) = \hat{H}\sin(\omega t)$, $\hat{H}$ being the amplitude and ω the frequency. During the experiment, one measures the magnetization m. If one starts with a system having $m = 0$, the magnetization will initially follow the dashed lines in Fig. 11.3. The shape of the hysteresis depends on the amplitude $\hat{H}$ of the external magnetic field. The area enclosed by the hysteresis grows with increasing amplitude $\hat{H}$. This can be used to *demagnetize* a probe, by starting with a large amplitude $\hat{H}$ and decreasing it gradually iteration by iteration, i. e., $\hat{H}$ will depend on the time as well. With $\hat{H}(t) \to 0$, the system will approach $m = 0$. In this way one has optimized the system, i. e., minimized the magnetization. The basic idea of varying external fields in a systematic way, with gradually decreasing amplitude of the field, to obtain an optimized system, can be generalized to other cases. This is called *hysteric optimization* [31]. Here, we will outline the method, a detailed pedagogical presentation can be found in Ref. [32].

As an example application, we will here consider the case of minimizing the energy of a spin glass, see Sec. 11.7 for more background on this model. To make the system suitable for hysteretic optimization, a term for a local magnetic field is added. The system consists of a set of interacting Ising spins $\sigma_i = \pm 1$, the Hamiltonian is given by:

$$\mathcal{H} = -\frac{1}{2}\sum_{i,j=1}^{N} J_{ij}\sigma_i\sigma_j - H\sum_{i=1}^{N}\xi_i\sigma_i, \tag{11.21}$$

where J_{ij} characterizes the interaction of spins i and j, H is the strength of the external field, and $\xi_i = \pm 1$ characterizes the local orientation of the external field at site i. For a

homogeneous field, $\xi_i = 1$ for all i. For the optimization, it will be useful to consider fields with random orientations at each spin position. First, the values ξ_i will be fixed during the optimization, later a procedure called *shake-up* is introduced, where the signs of some ξ_i are altered. Here the three-dimensional Edwards–Anderson (EA) spin glass, having $N = L^3$ spins, is treated. The interactions are between nearest neighbors, and they are drawn from a Gaussian distribution with zero mean and unit variance.

To perform a hysteresis, one has to measure the "magnetization". Instead of using the traditional magnetization, to account for the local field orientations ξ_i, here the following quantity is used:

$$m = \frac{1}{N} \sum_i \xi_i \sigma_i \,. \tag{11.22}$$

Hence, if $\xi_i = 1$ for sites, then this definition agrees with the usual definition of a magnetization.

The hysteresis loops are generated using a $T = 0$ dynamics. This means that all spins will be always aligned with their local effective field $v_i = w_i + H\xi_i$, $w_i = \sum_j J_{ij}\sigma_j$ being the local internal field. This means $\sigma_i v_i > 0$ always holds. One starts with a large value $H = H_0$ of the external field, such that all spins will be aligned with their local field, i. e., $\sigma_i = \xi_i$. Then the field strength H is reduced to the value $H = H_S$, where the first spin becomes locally unstable, i. e., it flips.[2] Since the dynamic is totally deterministic, one can pick the value of this field directly, by maximizing over all spins the value of the field where each spin becomes unstable. After the spin has been flipped, the local values of its neighbors are changed, hence some of its neighbors may flip as well. This generates an *avalanche* of spin flips, until all spins are again aligned with their effective local fields. This way of performing an hysteresis corresponds to an infinitely slowly change of the field, an *adiabatic* limit, because during an avalanche no change of the external field occurs. After an avalanche has been completed, other spins, which have become unstable independently at the same field value are treated in the same way.[3] Then the external field strength can be reduced further to the value where the next spin becomes unstable, again starting an avalanche of spin flips. Note that during the avalanches at different or even at the same values of the external field, any spin may flip several times back and forth, because the neighborhood may change several times. This process of decreasing H is repeated, until the final value $H_1 = -\gamma H_0$ is reached, $\gamma < 1$ being some predefined reduction factor. At this point, the field starts to increase again, until the value $H_2 = \gamma^2 H_0$ is reached, it then starts to decrease again, etc. This process is stopped, if no spin flip between two "turning fields" H_k and H_{k+1} has occurred.

In the left-half of Fig. 11.4 a hysteresis obtained for an $L = 10$ EA spin glass obtained for $\gamma = 0.9$, is shown. In the right-half of Fig. 11.4 the variation of the internal energy, i. e., the value of the first contribution of the Hamiltonian Eq. (11.21) is presented. One can see that the internal energy oscillates during the hysteresis iterations and decreases, on average, gradually. Hence, the hysteresis process indeed leads to low energies. For comparison, the true ground-state energy obtained with a genetic algorithm [33] is shown. One observes that the hysteric

[2] Hence it makes sense to choose directly $H_0 = H_S$.

[3] This may happen, in particular for systems with a high degeneracy, i. e., the $\pm J$ spin glass.

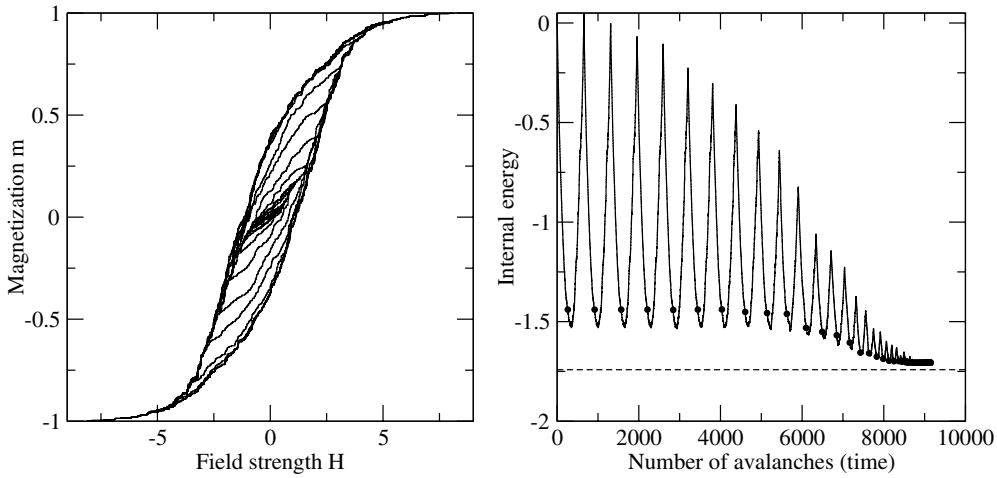


Figure 11.4: Magnetization curve (left) and the internal energy per spin during the demagnetization process (right) for an $N = 10^3$ three-dimensional Edwards–Anderson spin glass. Dots show values at zero field, dashed line marks the ground state energy.

optimization is able to approach the true ground-state energy very closely, but usually does not hit the actual ground states, this has also been seen in a recent study on random-field systems [34].

One way to improve the results is to use a larger reduction factor, e. g., $\gamma = 0.95$. One finds that the results somehow get better, but it does not pay to increase γ further towards 1, in contrast to simulated annealing (see Sec. 11.1), where it is guaranteed to find a ground state in the limit of infinite slow cooling. Instead, here it is more efficient to re-run the algorithm with a different random choice of the local-field orientations $\{\xi_i\}$, this is called a *shake-up*. It is not efficient to perform several independent shake-up runs, each starting with a locally stable configuration. Instead, the highest efficiency is obtained, when one continues from a previously found low-energy configuration. Also it is better to use a smaller maximum field strength $H_{\mathrm{shake}} << H_0$. Each shake-up is performed similarly to the initial hysteresis run: first the field is increased from 0 to H_{shake}, then the field is decreased down to $H = -\gamma H_{\mathrm{shake}}$, then up to $H = \gamma^2 H_{\mathrm{shake}}$, etc. Several shake-ups can be performed one after the other, where always the best configuration encountered so far is used as the starting configuration. A full cycle consisting of the initial demagnetization followed by several shake-ups is called *hysteric optimization* (HO). It turns out [31] that performing several dozen shake-ups allows us to come very close to the true ground-state energy of the EA spin glass.[4]

The reason why HO is not the most efficient method for the EA model, is probably that low-lying local optima differ from the true ground state by a flip of a whole connected group of spins. Within the demagnetization process only single-spin flips are used. Furthermore, for

[4]For the mean-field spin glass, where each spin interacts with each other one, even the true ground state was often found for more than five shake-ups.

the EA model, it turns out that the avalanches are typically very small.[5] This means that the hysteric optimization will be easily caught in local minima. As a consequence, it is favorable to combine HO with other techniques like genetic algorithms (see Sec. 11.3, a very efficient implementation for spin glasses is discussed in Refs [33, 35]) or renormalization-group based optimization [36, 37], which both allow for changes of the configurations involving a larger number of spins. Hence, HO can be seen as a building block within which are more complex optimization schemes, e. g., replacing simple local optimization techniques.

The HO scheme can be generalized to all kinds of optimization problems. For this purpose, one has to identify the possible configurations P as states of a physical system and use the cost function to be optimized as energy for the corresponding physical system, similar to simulated annealing discussed in Sec. 11.1.1. Also, one has to define a suitable dynamic, i. e., an elementary move within the configuration space. A suitable elementary move should not change the configurations too much. To make the system suitable for HO, one must add an extra term to the cost function, which accounts for the interaction with the external field H. The most simple choice is to add a term $Hd(P, R_\alpha)$, where $d(.,.)$ is a distance measure on the configuration space, and R_α is a predefined random *reference configuration*, corresponding to the random local orientations $\{\xi_i\}$ in the spin-glass case. The iterated hysteresis starts in a similar way to the spin-glass optimization discussed before. The initial configuration is the reference configuration, and one decreases the strength of H, while performing all occurring avalanches, until $H = 0$ is reached. In contrast to the spin-glass case, usually the system to be optimization does not exhibit a spin-flip symmetry. Thus, it does not make sense to go to a negative value of the field. Instead, a new random reference configuration is chosen, and the field is increased up to $H_1 = \gamma H_0$. Then the field is decreased down to $H = 0$, another new random reference configuration is chosen, and the field is increased again, now up to $H_2 = \gamma^2 H_0$. This process is continued, up to a largest value H_k such that no avalanche occurs in $H \in [0, H_k]$. Doing shake-ups is also straightforward, since one just has to increase the field up to $H_{\rm shake} < H_0$ (with a new random reference configuration), and continue as for the initial hysteresis, again slowly decreasing the amplitude of the varying external field.

In Ref. [32] the generalized HO has been applied to the traveling salesman problem TSP (see Chap. 1). For a system consisting of 100 cities, with a suitable choice for the local moves, the optimum tours were found by the HO algorithm. This shows that this optimization algorithm may be suitable for application to a broad range of problems.

11.3 Genetic algorithms

Genetic algorithms (GA) are fairly general optimization methods, and have been applied in numerous fields, see the end of this section for an incomplete list of applications in physics. For a detailed introduction, see, e. g., [38–40]. The basic observation, which led to the development of genetic algorithms [41], is that for millions of years a huge optimization program has been running: nature tries to adjust the way creatures are formed such that they have a

[5] In contrast to the mean-field model, where the avalanche size follows a power-law distribution.

high probability of survival in a hostile environment (unfortunately the hostile environment evolves as well). At the beginning, all animals and plants were very simple. Only by *reproduction* and *mutation* did the structure of the creatures become more and more complex, so that they adapted better and better to the requirements. This works, because of the presence of *selection*: individuals which are better equipped have a higher probability of staying alive ("survival of the fittest"). Hence, they can pass on the information about how they are built, i. e., their *genes*, to subsequent generations more often. By iterating this scheme, millions of times on a large *population* of individuals, on average, better and better creatures appear. Please note that this scheme is over simplified, because it neglects the influence of learning, society (which is itself determined somehow by the genes as well), etc.

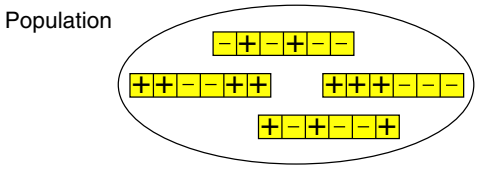


Figure 11.5: A population of individuals. Each individual is a collection of characteristic values, e. g., genes, parameters or positions. Here it is a vector of values $+/-$.

This simple principle can be transfered to other optimization problems. One is not necessarily interested in obtaining an optimum creature, but perhaps one would like to have a configuration of minimum energy (ground state) of a physical system, a motor with low energy consumption or a scheme to organize a company in an efficient way. Physical systems, motors or companies are not represented by a sequence of genes but are given through a configuration of particles or a vector of parameters. For the purpose of GAs, they will be denoted as *individuals* as well. The *population* (see Fig. 11.5) is again just a set of different individuals.

Now the different operations affecting the population will be considered. Mutations of genes correspond to random changes in the individual, e. g., displacing a particle or changing a parameter, see Fig. 11.6. The reproduction scheme, often called *crossover*, can be transfered in many different ways to GAs. The general principle is that one or several individuals (the *parents*) are taken, divided into small parts and reassembled in different ways to create new individuals, called *offspring* or *children*, see Fig. 11.7. For example a new particle configuration can be created by taking the positions of some particles from one configuration and the positions of the other particles from a second configuration.

The selection step can also be performed in many ways. First of all one has to evaluate the *fitness* of the individuals, i. e., to calculate the energy of the configurations or to calculate the efficiency of the motor with given parameters. In general, better individuals are kept, while worse individuals are thrown away, see Fig. 11.8. The details of the implementation depend on the problem. Sometimes the whole population is evaluated, and only the better half is kept. Or one could keep each individual with a probability which depends on the fitness, i. e., bad ones have a non-zero probability of being removed. Other selection schemes just compare the offspring with their parents and replace them if the offspring are better.

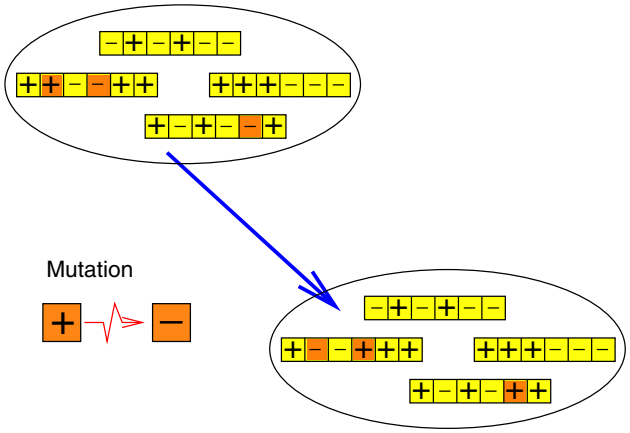


Figure 11.6: The effect of the mutation operation on a population. Individuals are randomly changed. Here, the values of the vectors are turned from $+$ to $-$ or vice versa with a given small probability. In the upper part the initial population is shown, in the lower part the result after the mutation has been carried through. The values which have been turned are highlighted.

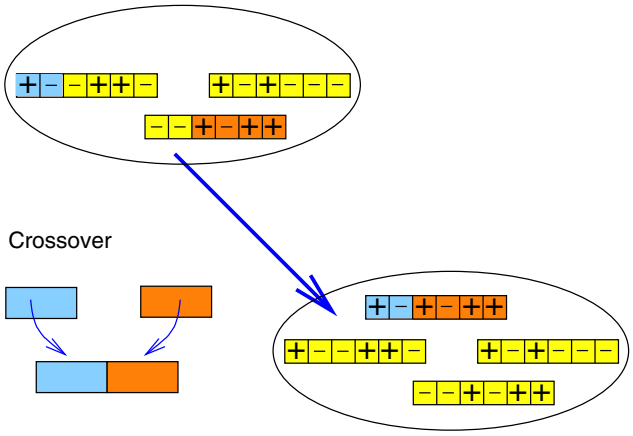


Figure 11.7: The crossover operation. Offspring are created by assembling parts from different individuals (parents). Here just one part from a $+/-$ vector and another part from a second vector are taken.

Another detail, which has to be considered first when thinking about implementing a GA, is the way the individuals are represented in a computer. In general, arbitrary data structures are possible, but they should facilitate the genetic operations such as mutation and crossover. In many cases a *binary representation* is chosen, i. e., each individual is stored via a string of bits. This is the standard case where one speaks of a "genetic algorithm". In other cases, where the data structures are more complicated, sometimes the notation "evolutionary program" is used. For simplicity, we just keep the expression "genetic algorithm", regardless of which data structure is chosen.

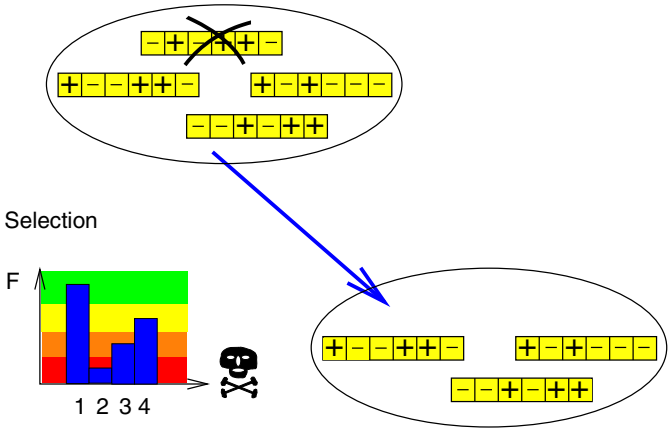


Figure 11.8: The effect of the selection operation on the population. The fitness F is evaluated, here for all four individuals. Individuals with a low fitness have a small probability of survival. In this case, individual two is removed from the population.

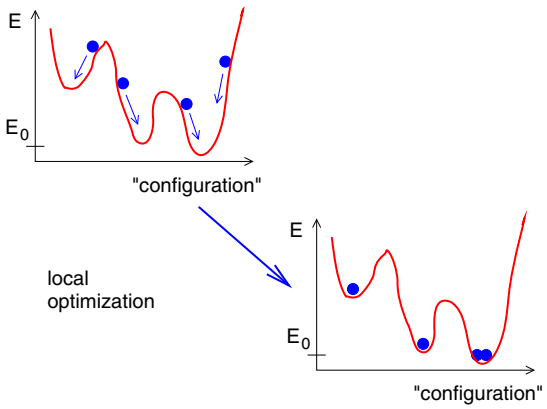


Figure 11.9: Local optimization. Here the population is shown in an energy landscape. Large energy means small fitness. This local optimization moves individuals to the next local optimum.

Quite frequently, the performance of a genetic algorithm can be increased by applying *local optimizations*. This means that individuals are taken and altered in a more or less deterministic way, such that their fitness is increased. Usually a local optimum is found which is close to the current individual, see Fig. 11.9. For example, when searching for a ground state, one could move particles in such a way that the energy is decreased, using the forces calculated from a given configuration. Whether and how local optimizations can be applied, depends strongly on the current problem.

We finish this section by summarizing a possible general structure of a GA. Please note that many different ways of implementing a genetic algorithm exist, in particular the order of the operations crossover, mutation, local optimization and selection, may vary. At the beginning the population is usually initialized randomly, M denotes its size and n_{R} the number of iterations.

algorithm genetic
begin
  Initialize population $x_1, \ldots, x_M$;
  for $t := 1$ **to** n_{R} **do**
  begin
    select parents $p_1, \ldots, p_k$;
    create offspring $c_1, \ldots, c_l$ via crossover;
    perform mutations;
    eventually perform local optimization;
    calculate fitness values;
    select individuals staying alive;
  end
end

Genetic algorithms are very general optimization schemes. It is possible to apply them to various problems appearing in science and everyday life. Furthermore, the programs are relatively easy to implement, no special mathematical knowledge is required. As a result an enormous number of applications have appeared during the last two decades. There are several specialized journals and conferences dedicated to this subject. When you search in the database INSPEC for articles in scientific journals which contain the term "genetic algorithm" in the abstract, you will obtain more than 15 000 references. On the other hand, applications in physics are less common, about several hundred publications can be found in INSPEC. Maybe this work will encourage more physicists to employ these methods. Recent applications include:

- Lattice gauge theory [42].
- Ground states of one-dimensional quantum systems [43, 44], see also Ref. [2].
- Analysis of X-ray data [45].
- Study of the structures of atomic clusters [46–48].
- Optimization of lasers [49]/ laser pulsed [50] and optical fibers [51].
- Examining nuclear reactions [52].
- Data assimilation in meteorology [53].
- Reconstructing geological structures from seismic measurements [54].
- Determination of orbital parameters of galaxies [55], see also Ref. [2].

However, one should mention that GAs have three major drawbacks. First, like the other methods presented in this chapter so far, the algorithm does not guarantee to find the exact op-

timum. Secondly, although the method has a general applicability, the actual implementation depends on the problem. Finally, you have to tune parameters like the size of the population or the mutation rate, to obtain a good performance. Very often the effort is worthwhile. The implementation is usually not very difficult. Some people might find it helpful to use the package *Genetic and Evolutionary Algorithm Toolbox* (GEATbx) [56], which is written for use in conjunction with the program Matlab [57].

11.4 Shortest paths and polymers in random media

Shortest-path algorithms are among the most basic polynomial-time running optimization algorithms in computer science. They have various applications in everyday life, e. g., routing programs will tell you the fastest train from your home town to the locations of the next interesting conferences. In molecular biology, shortest-path algorithms are used to compare protein sequences in large databases, this is the so-called *sequence alignment problem* [58–60]. Also in physics, shortest-path algorithms have many applications; as an example, we consider here the *directed polymer in a random medium* (DPRM).

The polymer in a random medium [2, 61] is essentially an elastic thread (a one-dimensional object – a line) lying in a rough landscape and searching for its lowest potential and elastic energy minimum. This problem can be mapped onto a graph with the random potential energies as distances and the shortest path as the ground-state configuration of the polymer. Two possible realizations for square lattices are shown in Fig. 11.10. For a path along the {10} orientation, the path is allowed to step forward, to the left or to the right, with an increased energy cost for motion to the left or right, which models the elasticity of the DPRM [62, 63]. An even simpler model is a path in the {11} orientation (see right part of Fig. 11.10). In this case, there is no explicit elasticity included, but the motion is restricted to the transverse direction, and it is believed that this constraint is sufficient to maintain the DPRM universality class.

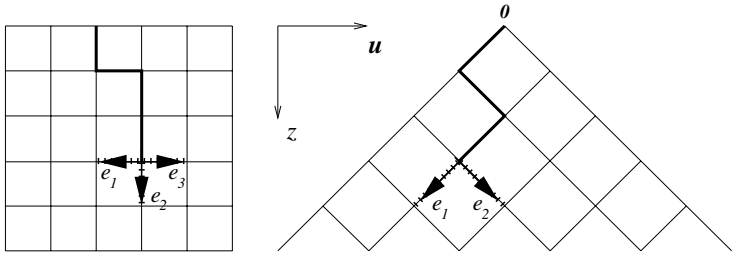


Figure 11.10: Models for a DPRM. Left: in the {10} orientation. Right: in the {11} orientation.

Mathematically, the DPRM is described by the following lattice Hamiltonian

$$H = \sum_{(ij)} e_{ij} \cdot x_{ij} \tag{11.23}$$

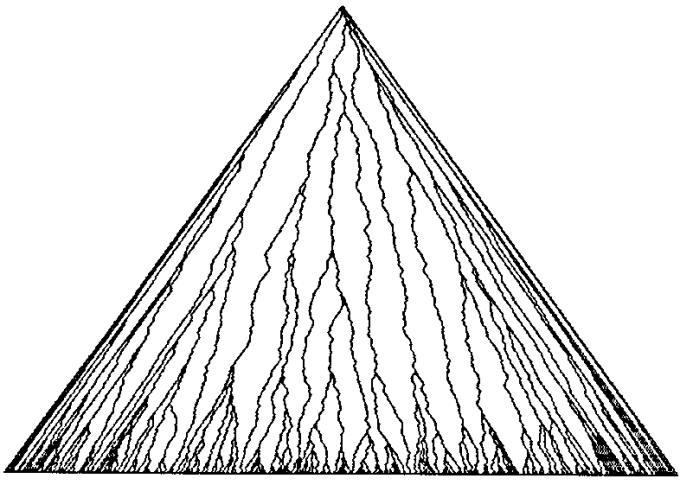


Figure 11.11: A collection of polymers of lowest energy directed along the diagonals of a square lattice with random bonds. Each polymer (crossing 500 bonds) has one end fixed to the apex of the triangle, the other to various points on its base, and finds the optimal path in between.

where the sum is over all bonds (ij) of the lattice and x_{ij} represents the DPRM configuration starting at one particular point s of the lattice and ending at another point t. It is $x_{ij} = 1$ if the DPRM passes through the bond (ij) and $x_{ij} = 0$ otherwise. Typically s is on one side of a lattice of linear size L and t on the opposite, and the ground state of (11.23) is the minimum energy path from s to t.

Interpreting the energies as distances (after making them all positive by adding a sufficiently large constant to all energies), and the lattice as a directed graph, this becomes a shortest-path problem that can be solved by using, for instance, Dijkstra's algorithm, which will be described below. In addition, owing to the directed structure of the lattice, one can compute the minimum energies of DP configurations ending at (or shortest paths leading to) specific target nodes t, recursively (this is the way in which Dijkstra's algorithm would proceed for this particular case) [64]. This is the same as the transfer-matrix algorithm, encountered in statistical mechanics [65]. It reduces, in the zero-temperature limit, to a simple recursion relation for the energies or distances from s to nodes t in the $(n+1)$th layer, once we know the shortest paths to the nth layer:

$$E_s^{(n+1)} = \min\{E_{s'}^{(n)} + e_{s's} | s' \in \ n'\text{th layer } \textit{and } s' \text{ nearest neighbor of } s\}. \quad (11.24)$$

As an example, we show in Fig. 11.11 a collection of such optimal paths to all base points of the triangle. In the following we will discuss one basic algorithm to find the shortest paths in a general graph.

The Dijkstra algorithm finds the shortest path from a source vertex s, in a directed graph $G = (V, E)$, to all other vertices. All distances c_{ij} in the graph have to be non-negative. The main idea is to store *distance labels* $d(i)$ for all vertices $i \in V$, where $d(i)$ contains the distance from s to i along the shortest known path so far. Also labels $pred(i)$ are kept, where for each vertex i its predecessor in the shortest path from s to i is stored.

The Dijkstra algorithm is called a *label-setting* algorithm, because in each iteration of the main loop, one distance label will be finally set. This is performed by portioning the set of vertices into two sets S and $\overline{S} = V \setminus S$, where S contains all those vertices, where the shortest path has already been found. Hence, initially S is empty, and in each iteration one vertex is moved from $\overline{S}$ to S. The set S can be seen as a *growth front* where, iteratively, vertices having the minimum distance are added.

Hence, to understand the algorithm, we assume that for all vertices belonging to the growth front S, the shortest path has already been found, and the distance labels of the members of the growth front are the shortest-path distances. For all vertices $i \in \overline{S}$ which are connected to the growth front, we can calculate, in the spirit of the transfer-matrix equation (11.24), the current optimal distances to i as $d(i) = \min_k\{d(k) + c_{ki}, k \in S, (k, i) \in E\}$. The main step is that now the vertex $i_{\min}$ which has minimum distance $d(i_{\min}) < d(i)\ (i \neq i_{\min})$ among the vertices in $\overline{S}$ is added to the growth front S. For this vertex, the shortest path is guaranteed to go only through the vertices of the current growth front, because all other vertices i outside the current growth front have larger shortest paths starting at s, than $i_{\min}$, hence the shortest path to $i_{\min}$ cannot go through any other vertex i. In other words, the vertices in S will always have shorter paths starting at s, than the vertices in $\overline{S}$. This is true at the beginning, because S is empty, and it is true during the computation, because the vertex with the shortest distance, among the vertices in $\overline{S}$, is always moved to S. This results in the following algorithm. Note that the minimum distances are always updated when a vertex is added to the growth front S.

algorithm Dijkstra($G, \{c_{ij}\}, s$)
begin
  $S := \emptyset$; $\overline{S} := V$;
  $d(s) := 0$; $pred(s) := 0$;
  for all vertices $i \neq s$ **do**
    $d(i) := \infty$; $pred(i) := 0$;
  while $\overline{S} \neq \emptyset$ **do**
  begin
    choose $i \in \overline{S}$ with minimum $d(i)$;
    $S := S \cup \{i\}$; $\overline{S} := \overline{S} \backslash \{i\}$;
    for all edges $(i, j) \in E$ **do**
      if $d(i) + c_{ij} < d(j)$ **then**
        $d(j) := d(i) + c_{ij}$; $pred(j) := i$;
  end
end

Example: Dijkstra algorithm

In Fig. 11.12, an example for the operation of the Dijkstra algorithm is shown. The shortest paths from the top vertex 1 to all other vertices are to be obtained. Initially the set S of permanently labeled vertices is empty, the distance label $d(1) = 0$ and all other distance labels are $d(i) = \infty$.

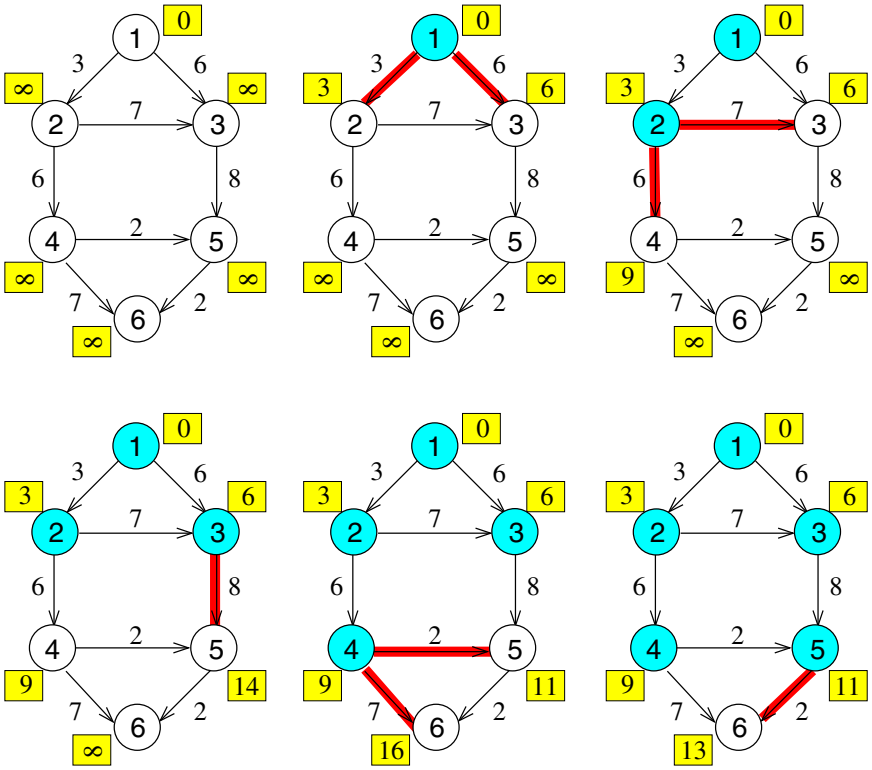


Figure 11.12: Example for the operation of Dijkstra's algorithm. The number in the boxes denote the distance labels $d(i)$. Open circles stand for vertices $i \in \overline{S}$, filled circles for permanently labeled vertices $i \in S$. The numbers on the edges are the distances c_{ij}. The edges to the neighbors considered in the current step are marked.

This means, in the first iteration, vertex 1 with $d(1) = 0$ will be chosen in $\overline{S}$ as having the minimum distance and moved to S. Within the loop, the distance labels of its neighbors, vertices 2 and 3, will be updated, because $d(1) + c_{12} = 0 + 3 < \infty$ and $d(1) + c_{13} = 0 + 6 < \infty$

In the next iteration, vertex 2 has the minimum distance $d(2) = 3$ among the vertices in $\overline{S}$. Its neighbors, vertices 3 and 4 are considered inside the loop. The distance label of vertex 3 will not be updated, because $d(2) + c_{23} = 10 > 6 = d(3)$. The distance label of vertex 4 will be updated to $d(4) := d(2) + c_{24} = 3 + 6 = 9$.

In the third iteration, vertex 3 will be chosen and the distance label of it neighbor, vertex 5, updated. In the fourth iteration, vertex 4 will be chosen. The distance label of its neighbor, vertex 5, will be updated again, because $d(4) + c_{45} = 11 < 14 = d(5)$. The distance label of vertex 6 will be updated the first time. In the fifth iteration, vertex 5 is chosen and vertex 6 updated again. In the last iteration vertex 6 is chosen. □

In the simplest implementation, where the minimum-distance vertex $i \in \overline{S}$ is chosen by running through all members of $\overline{S}$, the body of the loop takes $\mathcal{O}(|V| + |E|)$ steps. Since the loop is repeated $|V|$ times, this results in a total running time of $\mathcal{O}(|V|^2 + |V||E|)$. In the case, where the average connectivity is smaller than $\mathcal{O}(|V|)$, i. e., in the case of sparse graphs, one can do better. Then one can use a *priority queue* [66–68], most simply implemented by a *heap*, which allows one to select the minimum member among N elements in $\mathcal{O}(\log N)$ steps. Also, update operations like *insert* and *remove* can be done in $\mathcal{O}(\log N)$ steps. In this case, the running time of the Dijkstra algorithms comes down to $\mathcal{O}(|V| \log |V|)$.

As already mentioned, the Dijkstra algorithms works only for non-negative distances c_{ji}. There are other types of shortest-path algorithm [67, 68], namely, so-called *label-correcting* algorithms, where negative labels are also allowed. These algorithms converge only if the graph does not contain negative-distance cycles, because otherwise, when walking through the graph, one would decrease the distance with each cycle performed, i. e., the shortest paths would contain an infinite number of those cycles, leading to shortest paths having length $-\infty$. Detecting such negative cycles is important in the context of *minimum-cost flow* algorithms, which also have been applied to various physical problems like solid-on-solid models or vortex glasses. For details of these models and of the application of label-correcting algorithms in connection with minimum-cost flows, see Ref. [2].

11.5 Maximum flows and random-field systems

Random-field systems [69–72] and spin glasses [73–76] are two prototypical models with quenched disorder. The main constituents are localized magnetic spins with random interactions. These types of model exhibit almost all basic phenomena occurring in statistical physics of disordered systems, like complexity, glassiness and aging. The study of these systems, due to their behavior as distinct from the behavior of ordered systems, has led to many theoretical and experimental advances in physics. The latest development is the study of combinatorial optimization problems, as presented in this book.

Experimental realizations of random-field systems are *diluted antiferromagnets in a field* (DAFF) [77, 78]. One example is $Fe_xZn_{1-x}F_2$, where the variation of the local magnetic field on the Fe spins is caused by the varying concentration of Fe spins in the neighborhood due to the random replacement by non-magnetic Zn atoms (the pure system Fe_xF_2 is an antiferromagnet). Experiments on this system [79–81] show a thermodynamic phase transition from

an ordered low-temperature phase to a disordered high-temperature phase which is characterized by a logarithmic divergence of the specific heat, corresponding to the critical exponent[6] $\alpha = 0$. This phase transition is very different from ordered Ising systems, where $\alpha \approx 0.1$ has been found in three dimensions. We will show in this section that one can obtain these critical exponents, although measured in experiments and MC simulations at finite temperatures, from ground-state calculations of a simple model.

The most direct approach is to write down a Hamiltonian of an antiferromagnetic Ising system with non-magnetic impurities and an external homogeneous magnetic field. The behavior of true diluted antiferromagnets agrees very well with computer simulations [82–84] of this model. On the other hand, it is relatively hard to treat the model by means of an analytical approach [85]. As a consequence, *random-field systems* are usually studied instead, where the variation of the local magnetic field is not induced by the neighbors, but put in directly.

For the most simple random-field model, classical Ising spins are located on the sites of a regular lattice, having pair or multi-spin interactions. Whereas spin glasses (see next section) are characterized by random interactions, random-field systems exhibit ferromagnetic interactions only. The randomness and complexity of the model is due to local random magnetic fields. Here d-dimensional hypercubic lattices of Ising spins $\sigma_i = \pm 1$ are considered. For the *random-field Ising model* (RFIM), each spin interacts ferromagnetically with its nearest neighbors. A local random magnetic field B_i acts on each spin. The system has the following Hamilton function

$$H = -\sum_{\langle i,j \rangle} J_{ij}\sigma_i\sigma_j - \sum_i B_i\sigma_i \,. \tag{11.25}$$

The sum $\langle i, j \rangle$ runs over pairs of nearest neighbors. $J_{ij} > 0$ denotes the strength of the interaction, usually $J_{ij} = J$ is constant. Most of the time, quenched disorder realizations of the local magnetic fields $\{B_i\}$ distributed according a Gaussian distribution

$$p_G(B) = \frac{1}{\sqrt{2\pi h^2}} \exp\left(-\frac{B^2}{2h^2}\right) \tag{11.26}$$

with zero mean and width h are treated. Also a bimodal distribution, with $B_i = \pm h$ with equal probability, has often been considered.

Two-dimensional RFIM systems exhibit an ordered phase only at zero temperature [87]. For three or higher dimensions it has been shown rigorously [88] that an ordered ferromagnetic phase exists for low temperatures and low strengths, h, of the randomness. The phase diagram looks qualitatively like the mean-field phase diagram shown in Fig. 11.13. The transition between the ordered and the disordered phase is characterized by the above-mentioned critical exponents, which quantify the critical behavior of quantities like correlation length, specific heat, magnetic susceptibility or magnetization.

[6] Near a phase transition $T = T_c$, the behavior of the specific heat c can be described to first order by a $c \sim |T - T_c|^{-\alpha}$ behavior. In a similar way, critical exponents for other quantities like the correlation length ξ (exponent ν), magnetization m (exponent β) or susceptibility χ (exponent γ) can be defined.

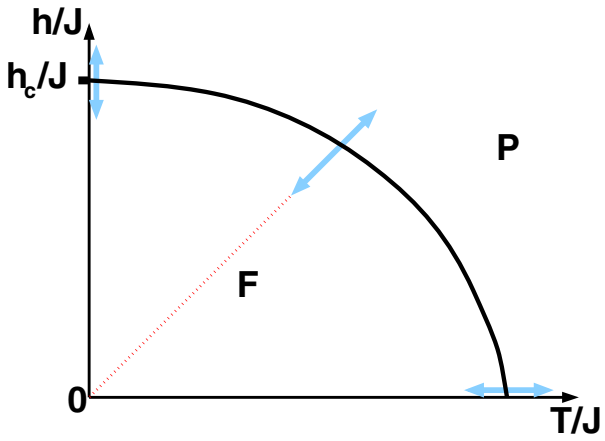


Figure 11.13: Phase diagram of RFIM with Gaussian distribution of the fields as obtained in mean-field theory [86], the transition is second order along the full line. F denotes the ferromagnetic phase, P the paramagnetic phase. For a bimodal distribution, there is a tricritical point on the line and the transition is first order above the tricritical point.

An analytical solution of the model has not been possible so far, for finite-dimensional systems. It has just been achieved for a mean-field model [86, 89], where every spin interacts with every other spin. For three-dimensional systems, computer simulations [90, 91] of small systems (up to $N = 16^3$ resp. $N = 32^3$ spins) have been performed to obtain the critical exponents. The values for most exponents are compatible with experimental results. Surprisingly, the specific heat did *not* show a logarithmic divergence, but a cusp, i. e., the value of the specific heat does not diverge but saturates at a finite value at the phase transition. This discrepancy, in comparison to the experiments, might be a result of the small system sizes or of equilibration problems within the Monte Carlo simulations.

Here ground-state calculations will turn out to be superior. Regarding the question of the values of the critical exponents, since the random-field fixed point of renormalization-group calculations is at $T = 0$ [92, 93], it can be assumed that the behavior at $T = 0$ is the same as that found for finite temperatures, at least near the fixed point. Thus the exponents for $T > 0$ can be obtained from classical ground-state calculations as well. As we will see below, the ground-state calculation for the RFIM belongs [4, 5, 94–96] to the class P, hence it is possible to treat much larger systems than is possible with MC simulations. Furthermore, since the ground-state algorithms are exact, one does not encounter equilibration problems.

Next, the mapping of the RFIM to *networks* is outlined and it is explained that the GS energy corresponds to the *maximum flow* through the network. This transformation was introduced by Picard and Ratliff in 1975 [94]. Furthermore it is described how the ground-state configuration can be obtained from the maximum flow. In the case where the ground state is degenerate, e. g., when a bimodal $\pm h$ distribution for the local fields is used, all ground states can be obtained

by a subsequent calculation [97–100] [7]. After this, the idea of an algorithm to obtain the maximum flow is presented briefly. Finally, it is explained how the different critical exponents can be obtained via the GS calculations.

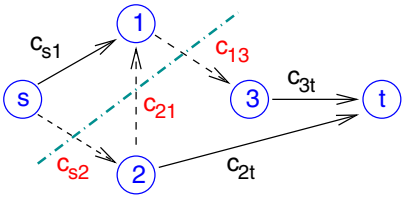


Figure 11.14: A sample network with 5 vertices $V = \{s, 1, 2, 3, t\}$, s denotes the source and t the sink. A cut $S = \{s, 1\}$, $\overline{S} = \{2, 3, t\}$ is indicated by the dashed-dotted line. The capacity of the cut is $C(S, \overline{S}) = c_{s2} + c_{13}$. The edge $(1, 2)$ does not contribute to the cut, because it is oriented in the opposite direction to it.

The algorithm is based on the notion of a network, which is a graph having, additionally, *capacities* assigned to the edges. Formally, a network is a four-tuple $N = (G, c, s, t)$, where $G = (V, E)$ is a directed graph which has $n + 2$ vertices, $c : V \times V \to \mathcal{R}_0^+$ are the capacities of the edges $(i, j) \in E$, with $c(i, j) = 0$ if $(i, j) \notin E$. The vertices $s, t \in V$ are the *source* and the *sink* of the network, respectively. For convenience, we use $V = \{0, 1, \ldots, n, n + 1\}$ where $s \equiv 0$ and $t \equiv n + 1$, and the notation $c_{ij} \equiv c(i, j)$. A sample network is shown in Fig. 11.14. The vertices in $V \setminus \{s, t\}$ are called *inner vertices*. Edges connecting only inner vertices are called *inner edges*.

A network can be interpreted as a system of streets connecting the source with the sink. The capacities c_{ij} indicate how much flow (e. g., number of cars) can run through a given route per time unit. Now, assume that the network should be divided into two pieces in such a way that the source and the sink are separated. You might think of the Austrian freeway system and a group of ecologists who like to prevent lorry drivers from taking the freeways to Italy[8]. Mathematically, this separation is a (s, t)-*cut* $(S, \overline{S})$ which has the following properties:

$$S \cup \overline{S} = V, \quad S \cap \overline{S} = \emptyset, \quad s \in S, \quad t \in \overline{S}. \tag{11.27}$$

A sample cut is shown in Fig. 11.14. Usually, we denote the elements of S *left* and the elements of $\overline{S}$ *right* of the cut.

Since the amount of work (or the number of ecologists), needed to block a street, grows with its capacity, the ecologists are interested in knowing the capacity of the streets they have to block. These are exactly the streets which go from S to $\overline{S}$, that means from left to right. We say these streets/edges *cross* the cut. Edges in the opposite direction cannot contribute additionally to a flow from the source to the sink, thus they are disregarded for the calculation

[7] There are other systems, where the ground-state calculation can be mapped onto maximum-flow problems in networks. Examples are interfaces in random elastic media [101, 102] and fracture surfaces in random-fuse networks [103].

[8] This happened in October 2002. It is not known whether mathematicians were involved in optimizing the blockade.

of the capacity. The sum of the capacities of the contributing edges is called the *capacity* $C(S,\overline{S})$ of the cut:

$$C(S,\overline{S}) = \sum_{i\in S, j\in\overline{S}} c_{ij}\,. \tag{11.28}$$

By introducing a binary vector $\underline{X} = (x_0, x_1, \ldots, x_n, x_{n+1})$ with $x_i = 1$ if $i \in S$ and $x_i = 0$ if $i \in \overline{S}$ one can rewrite the capacity of a cut as a quadratic function over the x_is via $C(\underline{X}) = \sum_{i<j\in V} c_{ij}x_i(1-x_j)$. Hence, it is immediately obvious that the energy of a RFIM Eq. (11.25) is equivalent to the capacity of a cut. To work out the details of the correspondence, one uses the fact that the source and the sink are fixed, i. e., $x_0 = 1$, $x_{n+1} = 0$, and that one has to map the 0/1-variables on spin values, i. e., $x_i = 0.5(\sigma_i + 1)$ for $i = 1, \ldots, n$. This results in a one-to-one correspondence of the interaction constants $\{J_{ij}\}$ and the values of the random-fields $\{B_i\}$ to the capacities $\{c_{ij}\}$. The capacities of the inner edges are determined directly by the interaction constants, while the capacities of the other edges depend also on the random fields. For details of the simple algebra see Ref. [2]. The final result is

$$\begin{aligned} C(\sigma_1,\ldots,\sigma_n) \;=\; & -\sum_{i<j}\frac{1}{4}(c_{ij}+c_{ji})\sigma_i\sigma_j \\ & +\sum_i\left(-\frac{1}{2}c_{0i}+\frac{1}{2}c_{i,n+1}+\frac{1}{4}\sum_j(c_{ij}-c_{ji})\right)\sigma_i \\ & +\frac{1}{4}\sum_{i<j}(c_{ij}+c_{ji})+\frac{1}{2}\sum_i(c_{0i}+c_{i,n+1})+c_{0,n+1}\,. \end{aligned} \tag{11.29}$$

Note that the sums run from 1 to n ($n =$ number of spins) and that the last line consists just of constants, which can be ignored when comparing with the Hamiltonian in Eq. (11.25). Hence, every configuration of the system corresponds to a cut in the network via $\sigma_i = 2x_i - 1$ and the energy $H(\sigma_1, \ldots, \sigma_n)$ is equal to the capacity $C(x_1, \ldots, x_n)$ of the corresponding cut.

Since we are interested in ground states, a minimum of the energy is to be obtained. Consequently, we are looking for a *minimum cut*, that is a cut of minimum capacity. Such a cut cannot be obtained directly, but is related to the flow going through the network in the following way. Each flow must pass the edges crossing an arbitrary cut, in particular the minimum cut. Therefore, the minimum cut capacity is an upper bound for the flow. On the other hand, it can be shown that the maximum flow which is possible is indeed given by the capacity of a minimum cut. The proof was found by Ford and Fulkerson in 1956 [104]. Versions of the proof which are more instructive can be found in Refs [105–107]. Along with the proof, a simple method for finding the maximum flow was introduced. Hence, after constructing an equivalent network, such a maximum-flow algorithm can be applied to find a ground state of a random-field Ising system. The Ford–Fulkerson algorithm is outlined below.

In addition to calculating the ground-state energy, one can also obtain the corresponding ground-state configurations. *One* single minimum cut, which represents one spin configuration of minimum energy, can be found in the following way, which is a special variant of

a breadth-first search. The basic idea is to start at the source and then to follow only edges where the flow is less than the capacity. Consequently, the spins visited are always on the left side of the minimum cut, since only edges which are satisfied can cross the minimum cut. After all vertices have been visited, which are accessible from the source in this way, the iteration stops. Then the spins which have been visited are left of the cut, so they are set to orientation $+1$. The remaining spins are set to orientation -1.

In this way only one ground state can be obtained, even if the system is degenerate. A simple method of finding *different* ground states in different runs of an algorithm has been presented in [98]. To actually find *all* different ground states one must find *all* existing minimum cuts. This can be achieved with a method explained in Ref. [97]. The result is a graph which describes all possible minimum cuts. For a Gaussian distribution of the random fields, the ground state is unique, the simple method, used to obtain the ground state configuration, is sufficient in this case.

Next, the basic idea of algorithms for calculating the maximum flow in a network is given. Mainly a description of the, historically, first method, which was invented by Ford and Fulkerson in 1956 [104], is presented here. Then we outline, how the extensions of the Ford–Fulkerson method, which are used in modern algorithms, work.

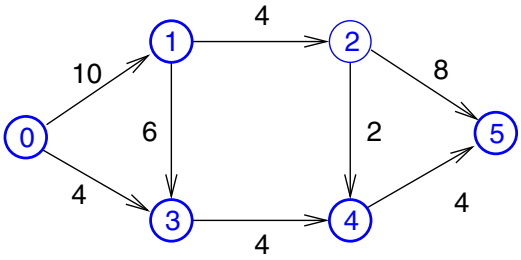


Figure 11.15: A small sample network. Vertex 0 is the source, vertex 5 the sink. The numbers close to the edges denote the edge capacities.

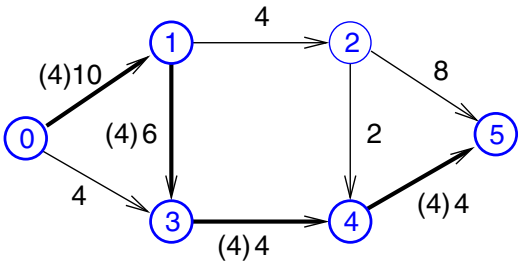


Figure 11.16: An augmenting path (bold edges) in the sample network. The numbers in brackets denote the flow values.

The basic idea of the Ford–Fulkerson algorithm is to start with an empty network, for an example see Fig. 11.15. Then the algorithm tries iteratively to push additional flow from the source to the sink. This is done by searching for paths along which the flow can be

increased, these are called *augmenting paths*. A sample of an augmenting path is shown in Fig. 11.16. The amount of flow, which can be additionally pushed along the augmenting path, is determined by the minimum *residual capacity*, i. e., the minimum difference between capacity and current flow. Note that for each single edge, sometimes it may be necessary to decrease the flow in order to increase the total flow. This case appears when a certain amount of flow is redirected within the graph, as shown in Fig. 11.17. If no augmenting path is found, the algorithm stops and the maximum flow has been found. [9]

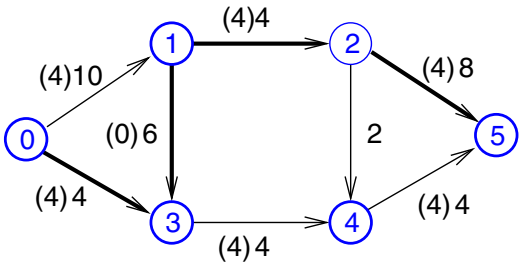


Figure 11.17: Another augmenting path (bold edges) in the sample network. Here the flow through the edge $(1, 3)$ is decreased to zero, hence the flow is redirected. No further flow increment is possible.

As mentioned, for each augmenting path, the edge with the smallest residual capacity $r_{\min}$ is a bottleneck. This means the flow along the path can be increased only by $r_{\min}$, which slows down the computation. Even worse, the running time of the Ford–Fulkerson algorithm is *not* bounded by a polynomial in the number of nodes $N = |V|$. The reason is that an "unfortunate" choice of the augmenting paths might lead to many iterations where, e. g., the flow is alternately increased through one path and then redirected through another path. An example is given in Ref. [2]. The running time of the Ford–Fulkerson algorithm is instead bounded by the capacity of the minimum cut, which itself is bounded by $|E|c_{\max}$, where $c_{\max}$ is the maximum occurring capacity.

This unfavorable behavior is avoided by an extension of the algorithm, which was presented by Edmonds and Karp in 1972 [109]. The basic idea is to choose an augmenting path, which has the shortest distance from the source to the sink, each edge contributes the distance one[10]. It can be shown that, indeed, this algorithm has a polynomial worst-case time complexity of $\Theta(|V||E|^2)$, which is in fact independent of the capacities of the edges.

Modern algorithms are even faster. The basic idea is that the flow is not augmented through one path, but through *many* paths at the same time, as proposed by Dinic [110]. This concept is used in Tarjan's *wave-algorithm* [111] and in so-called *push-relabel* algorithms [112–114]. The push-relabel method is implemented in the LEDA library [115], which is distributed commercially. A free implementation is included in the *Boost library* [116]. For the RFIM,

[9] The algorithm works for rational capacity values. The algorithm may not converge to the true maximum flow if the capacities are irrational, an example is given in Ref. [108].

[10] Another version of the algorithm always chooses the augmenting path with the maximum increment $r_{\min}$. This results in a time complexity of $\Theta(|E| \log c_{\max})$, where $c_{\max}$ is the maximum capacity of all edges.

the push-relabel method exhibits a running time, which grows like $\mathcal{O}(N^{4/3})$ with the system size [99, 117], i. e., only slightly stronger than linearly with the number of spins. Using these modern algorithms, system sizes like $N = 100^3$ are feasible on typical workstations having 500 MB of main memory.

As already mentioned, the critical exponents of the ferromagnet–paramagnet transition line can be obtained from ground-state calculations. This works by calculating ground states for several values h of the random-field strength and performing an average over the realizations, denoted by $[\ldots]_h$. Each realization is characterized by a set $\{\epsilon_i\}$ of random variables taken from a Gaussian distribution with zero mean and unit width, the random fields are $B_i = h\epsilon_i$. In this way one can directly measure the average absolute *magnetization* $m = \frac{1}{N}[|\sum_i \sigma_i|]_h$ and the *disconnected susceptibility* $\chi_d = \frac{1}{N}[(\sum_i \sigma_i)^2]$ as a function of the disorder strength h. This results in large magnetization for small disorder and a rapid decrease close to the transition h_c, where $h_c = 2.28(1)$ was found [118]. For larger disorder, the magnetization is zero. More results describing this transition, in particular critical exponents, can be found in the literature [95, 119–122]. Here, we will only show how the exponents concerning measurable quantities, which are not directly accessible, like (connected) susceptibility and specific heat, can nevertheless be obtained from GS calculations.

The susceptibility can be measured by considering the response to a small uniform external field H, i. e., one considers the Hamiltonian

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j - \sum_i B_i \sigma_i - H \sum_i \sigma_i \,. \tag{11.30}$$

For each given realization and value of h, one calculates the magnetization for few different small values of H. Near $H = 0$, the data points can be fitted very well with a parabola, the coefficient of the linear term gives the zero-field susceptibility $\chi = dm/dH|_{H=0}$. Results for the susceptibility and the corresponding critical exponent can be found in Ref. [123].

Finally, we consider the specific heat, which is usually obtained by measuring the fluctuations in the energy. This is not possible at $T = 0$, because there are no fluctuations in the energy. Going back to first principles, the specific-heat exponent is obtained from the singularity in the second derivative of the free energy with respect to temperature. More generally it is determined from the singularity obtained by varying a parameter which crosses the phase boundary from the paramagnetic phase to the ferromagnetic phase. From Fig. 11.13 we see that this can be conveniently accomplished by keeping the ratio of h/J to T/J fixed, i. e., by varying J. The first derivative of the free energy (per spin) F with respect to J, called here the "bond energy" E_J, is given by

$$E_J \equiv \frac{\partial F}{\partial J} = -\frac{1}{N} \sum_{\langle i,j \rangle} \langle \sigma_i \sigma_j \rangle, \tag{11.31}$$

where $\langle \cdots \rangle$ is a thermal average, and the sum is over nearest-neighbor pairs. Note that the thermal average is trivial at $T = 0$ in the case when the system is not degenerate. E_J has an energy-like singularity in the vicinity of the phase boundary and can be evaluated everywhere, also at $T = 0$.

Derivatives of E_J cannot be calculated analytically, hence a first-order finite difference must be applied to determine the derivative of E_J for ground states, numerically. Since this is a more accurate representation of the derivative at the midpoint of the interval than it is at either endpoint, the "specific heat", C, at $T = 0$ is defined to be

$$C\left(\frac{h_1 + h_2}{2}\right) = \frac{[E_J(h_1)]_h - [E_J(h_2)]_h}{h_1 - h_2}, \tag{11.32}$$

where h_1 and h_2 are two "close-by" values of h, i. e., here the derivative $\frac{\partial^2 F}{\partial J \partial h}$ is considered. It can be shown [123] that all second derivatives involving J and h, exhibit the same critical behavior. It should be mentioned that the final results of these GS calculations [123] are compatible with the MC simulations [90], i. e., the specific heat does *not* diverge at the phase transition.[11] This means the discrepancy in the experiments on the DAFF is still unexplained. Hence, it might be possible that the RFIM is indeed not a good model for the DAFF. Further studies on the DAFF itself, where the GS can be obtained by max-flow algorithms as well, might clarify this question in the future.

We have seen that one can calculate exact ground states of the RFIM in polynomial time, hence large systems can be treated. This allows us to study the ferromagnet–paramagnet transition occurring at low temperatures when varying the disorder strength (for dimension $d \geq 3$). It is possible to obtain thermodynamic quantities like magnetization, susceptibility or a specific-heat like quantity from the GS calculations, and study the corresponding critical exponents which describe the behavior of these quantities close to the transition. This means that the full critical behavior is accessible when using GS calculations, for much larger sizes and with a higher accuracy, compared with MC simulations.

11.6 Submodular functions and free energy of Potts model

The q-state Potts model [124] for integer values of q consists of N spins $\sigma_i \in \{1, \ldots, q\}$ living on the sites of an arbitrary undirected graph or lattice G, with the Hamiltonian

$$\mathcal{H} = -\sum_{ij} J_{ij}\delta(\sigma_i, \sigma_j) \tag{11.33}$$

where the sum runs over the edges $\{i, j\}$ of G, J_{ij} denotes the interaction between to spins i, j and δ is the Kronecker delta. It is a generalization of the Ising model, which is represented by the case $q = 2$.

Even the ordered Potts model ($J_{ij} = 1$ everywhere) is of profound interest, because, for dimensions d larger than one, it exhibits order–disorder phase transitions [125], which are of second order for q smaller than a critical value $q_c(d)$, while they are of first order for $q > q_c(d)$. It has been analytically proved [126] that $q_c(2) = 4$, but, e. g., for $d = 3$, the exact value of q_c is not known. Here, we will consider the ferromagnetic random-bond Potts model, i. e., the coupling J_{ij} are positive random values. Again performing an average over the disorder is accomplished by taking a sample of random instances according to some distribution.

[11] Note that both a logarithmic divergence and a cusp correspond to the critical exponent $\alpha = 0$.

Most cases studied in this book show how to obtain ground states or low-energy excited states using optimization algorithms. Here, we will consider an example where even the partition function, i. e., the full thermodynamic behavior at *any* temperature, can be calculated by an optimization algorithm in the limit $q \to \infty$, i. e., in the limit of an infinite number of states. This limit, although sounding a bit academic, may be of interest, because it is very likely that this model belongs to the same universality class as the random transverse quantum Ising chain. Here, we will concentrate on the mapping between the Potts model partition function, the optimization problem and the related mathematical background. For details on the optimization algorithms applied here, we will refer the reader to the literature.

To obtain the partition function Z via an optimization algorithm, one has to rewrite Z in the Fortuin–Kastellein representation [127] by starting at the standard definition:

$$\begin{aligned} Z &= \sum_{\{\sigma\}} \exp\left(-\mathcal{H}/T\right) \\ &= \sum_{\{\sigma\}} \exp\left(\sum_{ij} J_{ij}\delta(\sigma_i,\sigma_j)/T\right) \\ &= \sum_{\{\sigma\}} \prod_{ij} \exp\left(J_{ij}\delta(\sigma_i,\sigma_j)/T\right) . \end{aligned}$$

Since the Kronecker δ can take only the values 0 and 1, we can write in general $\exp(a\delta) = 1 + (\exp(a) - 1)\delta$. By defining $v_{ij} = \exp(J_{ij}/T) - 1$ we obtain

$$\begin{aligned} Z &= \sum_{\{\sigma\}} \prod_{ij} \left(1 + v_{ij}\delta(\sigma_i,\sigma_j)\right) \\ &= \sum_{\{\sigma\}} \left(1 + v_{i_1 j_1}\delta(\sigma_{i_1},\sigma_{j_1})\right)\left(1 + v_{i_2 j_2}\delta(\sigma_{i_2},\sigma_{j_2})\right)\ldots\left(1 + v_{i_m j_m}\delta(\sigma_{i_m},\sigma_{j_m})\right) , \end{aligned}$$

if we denote the set of edges of the graph G by $E = \{\{i_1, j_1\}, \{i_2, j_2\}, \ldots, \{i_m, j_m\}\}$, i. e., m is the number of edges of G. The product of the m factors $(1 + v_{ij}\delta(\sigma_i, \sigma_j))$ can be rewritten as a sum over 2^m addends, i. e., all possible combinations of products, where from each factor either the 1 or the $v_{ij}\delta(\sigma_i, \sigma_j)$ is taken. This is the same as summing over all 2^m possible subgraphs $G' = (V, E')$ of G, i. e., over all possible subsets $E' \subseteq E$ of edges. For each subgraph a product over all edges $\{i, j\}$ of $v_{ij}\delta(\sigma_i, \sigma_j)$ is taken, with the convention that the product over an empty set is one. Hence we get

$$Z = \sum_{\{\sigma\}} \sum_{G' \subseteq G} \prod_{\{ij\} \in E'} v_{ij}\delta(\sigma_i,\sigma_j).$$

Our last step will be to get rid of the sum $\sum_{\{\sigma\}}$. Only those factors $v_{ij}\delta(\sigma_i, \sigma_j)$ contribute, where $\sigma_i = \sigma_j$ for *all* edges of G'. Each subgraph G' can be decomposed into its connected components. By transitivity this means that only those factors $v_{ij}\delta(\sigma_i, \sigma_j)$ contribute, where all spins within a connected component have the same orientation. Since there are q possible orientations, each component contributes q times the product of all v_{ij} over all edges. In-

dependently connected components contribute independently q times, hence, when denoting by $c(G')$ the number of connected components of G', each graph G' contributes $q^{c(G')}$ the product over all edges of v_{ij}. Finally we obtain

$$Z = \sum_{G' \subseteq G} q^{c(G')} \prod_{\{ij\} \in E'} v_{ij} \,. \tag{11.34}$$

For large temperatures $T \to \infty$, the values of v_{ij} will be small. Therefore, graphs with a small number of edges contribute stronger to the sum. In this way Eq. (11.34) describes a high-temperature series expansion. Note that the parameter q appears as a simple number. This allows to extend the definition of the Potts model to non-integer values of q, or, as we will show below, to study the limit $q \to \infty$. Before doing this, we present as a simple example the four-spin Ising ferromagnet.

Example: Ising ferromagnet

We calculate the partition function of the four-spin Ising ($q = 2$) ferromagnet with nearest-neighbor interaction $J_{ij} = 1$, see Fig. 11.18, with the Hamiltonian

$$\mathcal{H} = -\sum_{\langle i,j \rangle} \delta(\sigma_i, \sigma_j). \tag{11.35}$$

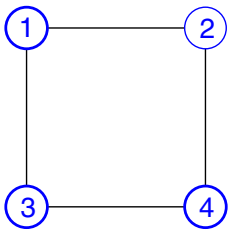


Figure 11.18: A tiny Ising ferromagnet with four spins and nearest-neighbor interactions.

For a direct summation one has principly to sum over 2^4 possible configurations. Since the Hamiltonian (11.35) is symmetric under inversion of all spins, we only have to consider the case, e. g., $\sigma_1 = 1$ and include a factor 2:

$$Z = 2 \sum_{\{\sigma_2,\sigma_3,\sigma_4\} = \{\pm 1\}^3} \exp(-\mathcal{H}/T) \,.$$

Now only four cases are possible:

- All spins are "up" ($+1$), i. e., bonds are satisfied, resulting in $\mathcal{H} = 0$.
- One spin is "down" (-1), the other three "up", i. e., two bonds are satisfied and two unsatisfied, i. e., $\mathcal{H} = 2$. Since either $\sigma_2 = -1$ or $\sigma_3 = -1$ or $\sigma_4 = -1$, this occurs three times.

- Two spins are down, two up. Either the two down spins are neighbors, then $\mathcal{H} = 2$ (two cases), or they are not neighbors, then $\mathcal{H} = 4$ (one case).
- Three spins are down, only σ_1 is up. Hence $\mathcal{H} = 2$ again.

. We obtain

$$Z = 2\left(e^0 + 3e^{-2/T} + (2e^{-2/T} + e^{-4/T}) + e^{-2/T}\right) \tag{11.36}$$

$$= 2(1 + 6e^{-2/T} + e^{-4/T})\,. \tag{11.37}$$

On the other hand, when using Eq. (11.34), we have to sum over all possible subsets of edges of the graph in Fig. 11.18, see Fig. 11.19.

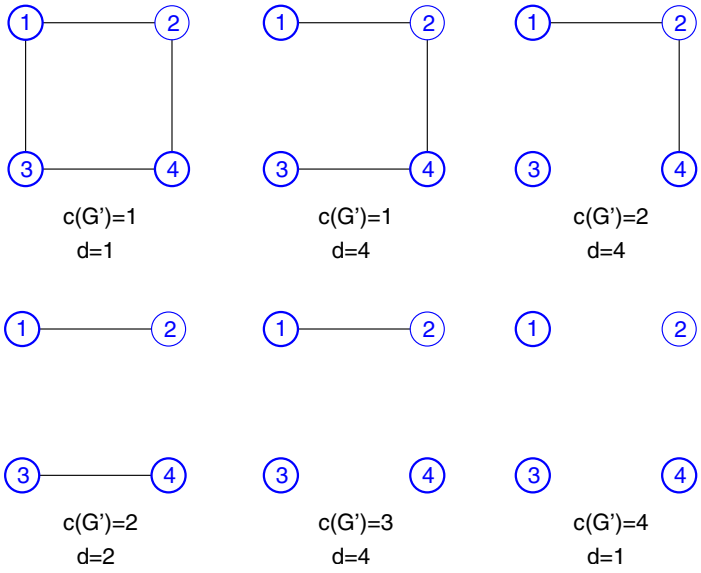


Figure 11.19: All six possible types of subgraphs G' of the four edge graph. $c(G')$ denotes the number of components and d the degeneracy, i. e., how often this type of subgraph exists.

We obtain, using $v = e^{1/T} - 1$

$$Z = q^1v^4 + 4q^1v^3 + 4q^2v^2 + 2q^2v^2 + 4q^3v + q^4$$

$$= 2v^4 + 8v^3 + 24v^2 + 32v + 16\,. \tag{11.38}$$

Note that in the case $T \to \infty$, only the graph with no edges and four components contributes, i. e., $Z \to 2^4 \prod 1 = 16$, which is also the result one gets when inspecting Eq. (11.37) in the same limit. By evaluating all powers of v in Eq. (11.38) you will see that indeed the same formula as in Eq. (11.37) results. □

By introducing [128] w_{ij} via $v_{ij} = q^{w_{ij}}$, we can rewrite (11.34) as

$$\begin{aligned} Z &= \sum_{G' \subseteq G} q^{c(G')} \prod_{\{ij\} \in E'} q^{w_{ij}} \\ &= \sum_{G' \subseteq G} q^{c(G') + \sum_{\{ij\} \in E'} w_{ij}} \\ &= \sum_{G' \subseteq G} q^{-f(G')} \end{aligned}$$

with

$$f(G') \equiv -c(G') - \sum_{\{ij\} \in E'} w_{ij}. \tag{11.39}$$

For $q \to \infty$ only subgraphs minimizing $f(G)$ will contribute to the partition function, hence obtaining the partition function in this limit is equivalent to an optimization problem. This minimization problem can be solved in polynomial time, because f has a particular property, it is a *submodular function*, which will be introduced now.

First, we define *set functions*. Let V be any set and $2^V \equiv \{A | A \subseteq V\}$ the set of all subsets. Any function $f : 2^V \to \mathbb{R}$ is a set function. Here, only rational-valued set functions are considered. A set function is called *submodular* [129], if for all subsets $A, B \subseteq V$:

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B)\,. \tag{11.40}$$

One can show [129] that a function is submodular iff for all subsets $S \subseteq R \subseteq V$ and for all elements $x \in V$:

$$f(S \cup \{x\}) - f(S) \geq f(R \cup \{x\}) - f(R). \tag{11.41}$$

This means that adding an element to a smaller set has "greater effect" than adding to a larger set.

Example: Simple submodular functions

If $f(V) = |V|$, i. e., f counts the number of elements of a set, then even equality holds in Eq. (11.40). In this case f is called a *modular* function.

Let V be a set, L an arbitrary set of *labels* and $l : V \to L$ a labeling of the elements $x \in V$. Let $c : 2^V \to \mathbb{N}$ a function assigning each subset $A \subseteq V$ the *number* of distinct labels in V. It is intuitively clear that the function c fulfils Eq. (11.41) because if adding an element x to the large set increases the number of labels (by one), then the number of labels will go up by one for all subsets as well. But if the number of labels goes up for a subset, then it does not necessarily go up for the large set, since the label might already be contained in the large set without adding x. □

The function f from Eq. (11.39) is also a submodular function. This function is defined on subsets of edges. To see the submodularity, we inspect the two parts of f independently.

- Let $E' \subseteq E'' \subseteq E$ be two sets of edges and $e = \{i,j\} \in E$ be any edge. Then we have $-c(E' \cup \{e\}) + c(E') \geq -c(E'' \cup \{e\}) + c(E'')$ which can be verified by considering independently the three possible cases $(e \in E', e \in E'')$, $(e \notin E', e \in E'')$ and $(e \notin E', e \notin E'')$. Hence, $-c$ is a submodular function.
- For any two subsets of edges E', E'' we see immediately that for $w(G) = \sum_{\{i,j\} \in E} w_{\{i,j\}}$ the equality $w(E') + w(E'') = w(E' \cap E'') + w(E' \cup E'')$, holds, i. e., $-w$ is submodular (since equality holds, it is even a modular function).

To find extrema of set functions may be extremely tedious, since the number of possible subsets of V is $2^{|V|}$, i. e., exponentially in the cardinality of V. Fortunately, it has been proved for rational-valued submodular functions that a polynomial algorithm in $|V|$ exists [130]. The algorithm given in that paper was not very efficient. Later, Schrijver [131] and Iwata–Fleischer–Fujishige [132] discovered independently an efficient combinatorial algorithm. For the details about these algorithms, we refer the reader to the literature.

For the special case treated here, i. e., the minimization of the function $f(G)$ when calculating the partition function of the Potts model, an algorithm which is even faster and simpler to implement exists [133, 134]. It works by starting with an empty subgraph and adding vertex after vertex to the subgraph, such that optimality is always kept. In each iteration, the optimum is found by a maximum-flow calculation, as described in Sec. 11.5. For details of the full algorithm see Ref. [134]. This algorithm exhibits a running time, which empirically was found to increase as $t \sim N^{2.4}$ with N the number of vertices of the system. Using that algorithm, partition functions for two-dimensional random-bond Potts models of size $N = 512 \times 512$ vertices could be calculated[12], i. e., the function f was optimized over $2^{2\times512\times512} \approx 2.6 \times 10^{157827}$ possible subsets of edges!

11.7 Matchings and spin glasses

In this section we are dealing with spin glasses, which are models of disordered magnetic alloys. A quick introduction has already been given in Chap. 1. Similar to the RFIMs, which were covered in Sec. 11.5, spin glasses consist of spins interacting with each other. However, their behavior is more complicated. Owing to the existence of competing interactions, they exhibit ordered phases, but without spatial order of the orientations of the spins. From the computational point of view, spin glasses are very interesting as well, because the ground-state calculation is NP-hard [135]. For these and other reasons, spin glasses have been among the central topics of research in material science and statistical physics during the last two decades. This can be seen from the fact that almost all heuristic optimization methods invented so far have been tested on spin glasses.

[12] Each bond had weight $w_{ij} = 1/6$ or $w_{ij=}5/6$, each with probability $1/2$. This is right at a critical point.

Introductions to spin glasses can be found in Refs [73–76]. Recent developments are covered in Ref. [72]. A suitable theoretical model describing spin glasses consists of N Ising spins $\sigma_i = \pm 1$ placed on the regular sites of a d-dimensional lattice with linear extension L, e. g., quadratic ($N = L^2$) or cubic ($N = L^3$). The spins interact ferromagnetically or antiferromagnetically with their neighbors. A small example is shown in Fig. 11.20. The Hamiltonian is given by

$$\mathcal{H} \equiv -\sum_{\langle i,j \rangle} J_{ij}\sigma_i\sigma_j - B\sum_i \sigma_i. \tag{11.42}$$

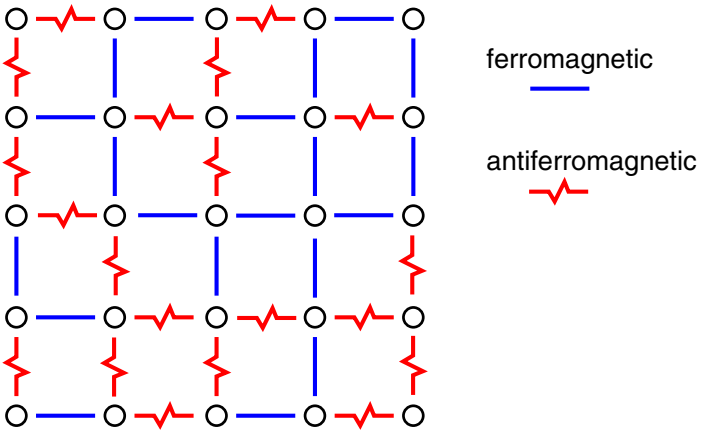


Figure 11.20: A two-dimensional spin glass with bond disorder. Spins are placed on the sites of a regular grid. They interact with their neighbors, the interaction is random, either ferromagnetic (straight lines) or antiferromagnetic (jagged lines).

The sum $\langle i, j \rangle$ runs over all pairs of nearest neighbors and J_{ij} denotes the strength of the bond connecting spins i and j. $J_{ij} > 0$ describes a ferromagnetic interaction, while $J_{ji} < 0$ an antiferromagnetic interaction. The last term describes the interaction with an external field B, here we will concentrate on the case $B = 0$. This kind of model was introduced by Edwards and Anderson [136] in 1975, usually it is called the EA model. It has a broad range of applications. Models involving similar energy formulae have been developed, e. g., for representing neural networks [137], social systems [138–140] or stock markets [141].

For each realization of the disorder, the values J_{ij} of the bonds are drawn according to a given probability distribution. The Gaussian distribution and the bimodal $\pm J$ distribution are very common and have the following probability densities:

$$p_G(J) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{J^2}{2}\right) \tag{11.43}$$

$$p_{\pm J}(J) = 0.5\delta(J-1) + 0.5\delta(J+1)\,. \tag{11.44}$$

Once the values of the bonds are fixed for a realization, they keep their values throughout the whole calculation or simulation, known as *quenched disorder*. Since the system is random

itself, to calculate physical quantities like magnetization or energy, one must perform not only a thermal average but also an average over different realizations of the disorder.

The main ingredients constituting a spin glass are: *mixed signs of the interactions* and *disorder*. As a consequence, there are inevitably spins which cannot fulfil all constraints imposed by their neighbors and the connecting bonds, i. e., there will be some ferromagnetic bonds connecting antiparallel spins and vice versa. It is therefore not possible to *satisfy* all bonds. This situation is called *frustration*, [142]. In Fig. 11.21 an example of a small frustrated system is shown.

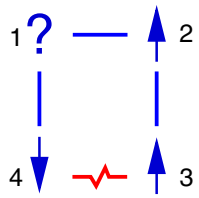


Figure 11.21: A frustrated plaquette at lowest energy. No matter which orientation spin 1 chooses, one of the bonds connecting it to its neighbors is not satisfied. Bonds 2–3 and 3–4 are satisfied.

Computer simulations of the EA model [71, 73] reproduce the main results found in experiments. a peak in susceptibility at a low temperature T_c, indicating a phase transition to an ordered phase, a smooth behavior of the specific heat around this temperature and frozen configurations of the spins below T_c. This means that the model is quite successful and can be used to gain a full understanding of the low-temperature frozen phase, which is still lacking despite huge efforts during the last three decades.

Here, we will now concentrate on two-dimensional Ising spin glasses, where it is now widely accepted that *no ordered phase* for finite temperatures exists [143–147]. For this special case, ground states can be calculated efficiently using a mapping to a matching problem (see Chap. 3). We will outline how the mapping works and concentrate particularly on how the ground-state algorithms can also be used to calculate excited states in a controlled manner. This allows us to verify some predictions made by using theoretical approaches.

Here, just the basic idea of the mapping to a matching algorithm will be explained. For the details, see Refs [148–150]. The method works for spin glasses which are planar graphs, e. g., for spin glasses with periodic boundary conditions in, at most, one direction and without external magnetic fields. On the left part of Fig. 11.22 a small two-dimensional system with open boundary conditions is shown. All spins are assumed to be "up", hence all antiferromagnetic bonds are not satisfied. If one draws a dotted line perpendicular to all unsatisfied bonds, one ends up with the situation shown in the figure: all dotted lines start or end at frustrated plaquettes and each frustrated plaquette is connected to exactly one other frustrated plaquette. Each pair of plaquettes is then said to be *matched*. Now, one can consider the frustrated plaquettes as the vertices and all possible pairs of connections as the edges of a (dual) graph. The dotted lines are selected from the edges connecting the vertices and are called a *perfect* matching, since *all* plaquettes are matched. One can assign to the edges in the dual graph,

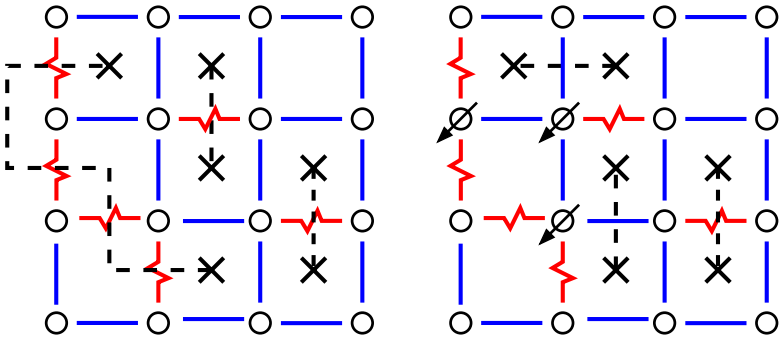


Figure 11.22: A two-dimensional spin glass with all spins up (left, up spins not shown). Straight lines are ferromagnetic, jagged lines antiferromagnetic, bonds. The dotted lines connect frustrated plaquettes (crosses). The bonds crossed by the dotted lines are unsatisfied. In the right part the GS with three spins pointing down (all others up), corresponding to a minimum number of unsatisfied bonds, is shown.

weights, which are equal to the sum of the absolute values of the bonds crossed by the dotted lines, i. e., of the unsatisfied interactions. The weight Λ of the matching is defined as the sum of the weights of the edges contained in the matching. Since Λ measures the unsatisfied bonds, the energy of the configuration is given by $E = -\sum_{\langle i,j\rangle} |J_{ij}| + 2\Lambda$. Note that this holds for *any* configuration of the spins, since a corresponding matching always exists. Obtaining a GS means minimizing the total weight of the broken bonds (see right panel of Fig. 11.22), so one is looking for a *minimum-weight perfect matching*. This problem is solvable in polynomial time. The algorithms for minimum-weight perfect matchings [151, 152] are among the most complicated algorithms for polynomial problems. Fortunately, the LEDA library offers a very efficient implementation [115], except that it consumes a lot of memory, which limits the size of the systems to about $N = 500^2$ on a typical 500 MB workstation.

As already indicated at the beginning of this chapter, not only obtaining true ground states is interesting, but the calculation of excited states using ground-state algorithms is one major task in this field. One possible general approach consists of these three steps:

1. Calculate the GS $\{\sigma_i^{(0)}\}$ of a given realization and the GS energy E_0.
2. Modify some of the couplings so that the GS is changed.
3. Calculate the GS $\{\sigma_i^{(m)}\}$ of the modified system, which is usually a low-lying excited state of the original realization. The ground-state energy of the modified system is denoted by $E_0^{(m)}$.

Here, we will consider two types of excites state, *domain walls* and *droplets*, which play a major role in the theoretical study of spin glasses. Other types of excitation, which can be generated using ground-state algorithms in the same spirit, are discussed in the literature [153–155]

A domain wall spanning the whole system can be generated, e. g., by switching the boundary conditions from periodic to antiperiodic in one direction. This is done by taking one column of bonds, e. g., those which "wrap around the system boundaries", and inverting the sign of all bonds in this column, see Fig. 11.23.

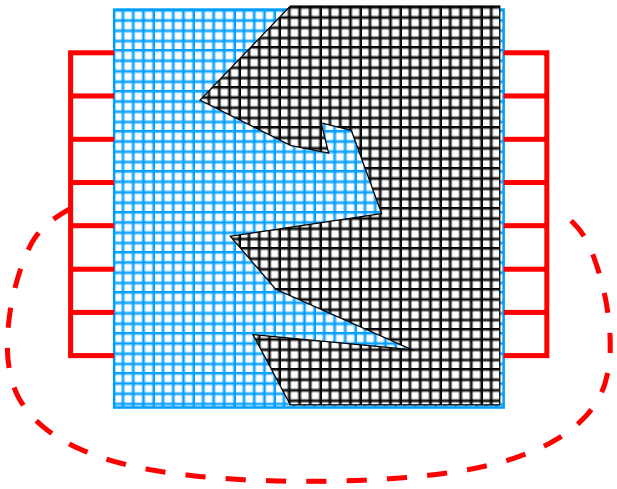


Figure 11.23: Method used to generate domain walls. After calculating the GS, the signs of all bonds connecting the pairs of spins in the first and last column (represented here by the thick broken line) are inverted. In this way two domains are generated; one, where the spins still have their GS orientations (indicated by the dark area); another, where the spins have opposite sign with respect to the GS (light area).

After calculating the GS with the modified system, the pairs of spins touching the modified bonds will have a relative orientation to each other, which is different from the GS obtained with the original bonds previously. In this way two domains are generated; one, where the spins have their GS orientations; another, where the spins have opposite sign with respect to the GS. The two domains are separated by the domain wall, which will adjust in a manner that the total energy is minimized. The GS energy of the modified system is locally everywhere the same as in $\{\sigma_i^{(0)}\}$, except at the domain wall. Hence, when calculating the energy difference between the GS energies E_0 of the original system and E_0^m of the modified system, one obtains exactly the energy of the domain wall: $\Delta E \equiv E_0^m - E_0$.

What can we learn from the behavior of the domain-wall energy? We will see that this approach allows us to determine whether an ordered phase can exist at low temperature. The basic idea is that the inversion of the bonds in one column corresponds roughly to a real-space renormalization [156, 157] of the system [158]. This means that one takes blocks of b^d spins, and replaces them by one *block spin*. One would like the renormalized system to be described by the same type of interactions as the original system. This means that the interactions between all members of two different blocks spins are replaced by just one pair interaction between the two block spins. In general, one can determine the value of the interac-

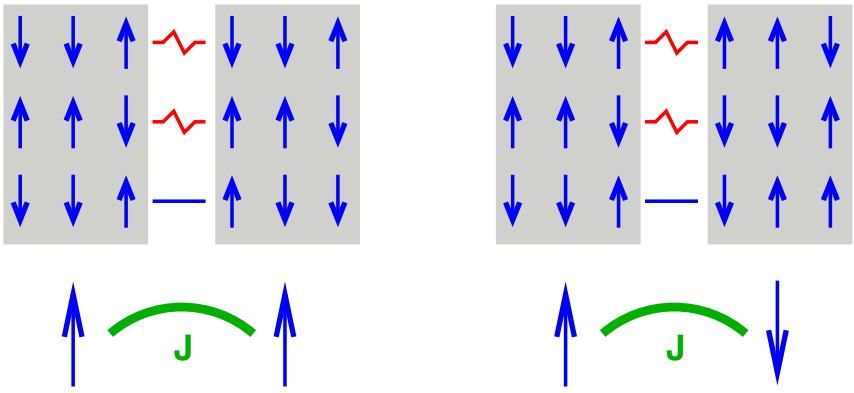


Figure 11.24: Groups of b^d spins are grouped together in *block spins*, here $b = 3$ and $d = 2$. One can determine the value of the effective coupling J between the block spins by calculating twice, first with both block spins parallel ($E_1 = -JS_1S_2 = -J$), then with both block spins antiparallel ($E_2 = +J$), hence $J = (E_2 - E_1)/2$.

tion between two spins by calculating the energy of the system, flipping one spin, calculating again the energy and taking (half) the difference between the two energies. This flipping of one block spin, corresponds to switching the boundary conditions from periodic to antiperiodic [158]. The only difference is that now the block size is of the order of the full system size, i. e., $b \sim L$, and that the system consists of just two block spins. In this case, it is not *a priori* defined, which spins belong to which block, because the domain wall will be the boundary between the two block spins. Hence, the block spins do not have hypercubic shape, but the spirit of the procedure of generating domain walls is similar to the renormalization approach.

Since the original system has quenched disorder, this will be the same for the renormalized system as well, but the distribution might be different. When averaging over many different realizations of the disorder, one will obtain distributions $P_L(\Delta E)$ of domain-wall energies, i. e., renormalized energies. Going to larger and larger system sizes L when calculating the domain-wall energy, corresponds to taking larger and larger block spins. What will we learn from the behavior of $P_L(\Delta E)$? First, we consider the first moment, i. e., the average. Switching all bonds in one column sometimes decreases the ground-state energy with respect to the original system, but sometimes the energy is increased. Since no case is particularly favored, the distributions $P_L(\Delta)$ will be symmetric with mean zero for all sizes L, so this does not give any information about the coupling of the block spins. Instead, a measure for the strength of the interaction distribution will be the width of the distribution, or, equivalently, the behavior of the absolute value $|\Delta E|$. From simple scaling arguments [158, 159] and in numerical experiments one finds a power-law behavior $|\Delta E| \sim L^\theta$, θ the so-called *stiffness exponent*. If $\theta < 0$, then the interactions become weaker with growing block size. This means, in the limit of infinitely large blocks, the block spins are no longer coupled, hence the system behaves paramagnetically at any finite temperature. This is indeed the case for the $d = 2$ EA model with Gaussian distribution of the interactions,

where $\theta \approx -0.28$ was found [143, 145, 147, 158, 160]. For larger-dimensional spin glasses, where no polynomially-running exact algorithms exist, positive values have been obtained, e. g., $\theta(d = 3) \approx 0.19$ [158, 161, 162] resp. $\theta(d = 4) \approx 0.65$ [163, 164]. For these systems, the couplings become stronger under renormalization, hence an ordered phase can exist at low temperatures, until entropic effects become too strong at higher temperature. Hence, we have seen that the calculation of domain walls via optimization algorithms allows us to determine whether an ordered phase can exist at low temperatures. Now we turn to the second kind of excitation discussed here, i. e., droplets.

Droplets are connected clusters of spins, which are reversed with respect to the GS. Droplets of minimum free energy with respect to a given range of sizes, i. e., having a length scale l, are a fundamental entity in one certain analytical approach to spin glasses, called the droplet scaling theory [159, 165, 166]. One central assumption of this theory is that the free energy ΔF of typical droplets scales like $\Delta F \sim l^{\theta'}$ with the droplet size, θ' being a characteristic droplet exponent. A prediction of the droplet scaling theory is that the exponents describing droplets and domain walls are the same, i. e., $\theta' = \theta$. For $T = 0$, the minimum free energy requirement translates to minimum energy conditions. This allows us to verify this and other predictions of the droplet scaling theory by GS calculations.

A traditional approach of generating droplets was used by Kawashima and Aoki within a Monte Carlo simulation [167]. They first calculated the GS heuristically (see, e. g., Sec. 11.1). Then they recalculated the GS with the constraints that the spins on the boundary keep their GS orientations, while a central spin is flipped. Fixing the spins can be achieved by introducing strong local fields acting on the spins, with the direction of the fields chosen to point along the desired direction, hence one has to apply a Hamiltonian with a linear field term.[13] This generates droplets of the order of the system size, i. e., $l \sim L$. Using this approach, small systems up to $L = 50$ with a Gaussian distribution of the interactions could be studied, and a scaling $\Delta E \sim L^{\theta'}$ of the droplet energy with $\theta' = -0.45(1)$ was found, which is significantly different from $\theta \approx -0.28$.

Since the range of system sizes in the work of Kawashima and Aoki is limited to $L \leq 50$, their result might have been an artifact of too-small sizes. Using matching algorithms, one can study much larger sizes. Unfortunately, it is not possible to treat local magnetic fields, i. e., one cannot fix a spin in some orientation. But it is possible to mimic this kind of generation of droplets in the following way. After obtaining the GS, several *hard bonds* are introduced. A hard bond is a bond with a high value of the absolute strength (e. g., $J_{\mathrm{big}} = 2N \times \max_{\langle i,j \rangle}\{J_{ij}\}$). The strength of a hard bond is so large such that the bond will be satisfied in all subsequent GS calculations. Most of the hard bonds are used to ensure that neighboring spins i_0, j_0 have the same relative orientation as in the ground state, i. e., one replaces bond $J_{i_0 j_0}$ by a bond with the value $J'_{i_0 j_0} = J_{\mathrm{big}} \sigma^{(0)}_{i_0} \sigma^{(0)}_{j_0}$. An *inverted* hard bond has the opposite sign, i. e., it forces two neighboring spins to take orientations, which are different relatively to the ground state orientations. This means in a GS calculation with the inverted hard bond, exactly one of the two spins will flip with respect to $\{\sigma^{(0)}_i\}$, the other spin will keep its previous GS orientation.

[13] Alternatively, one can just ignore the fixed spins in the second GS calculation and leave their current orientations. Then the fixed spins act as local fields to their neighbors.

Note that the subsystem of all hard bonds together must not exhibit frustration, because no hard bond can be broken when a new GS is calculated.

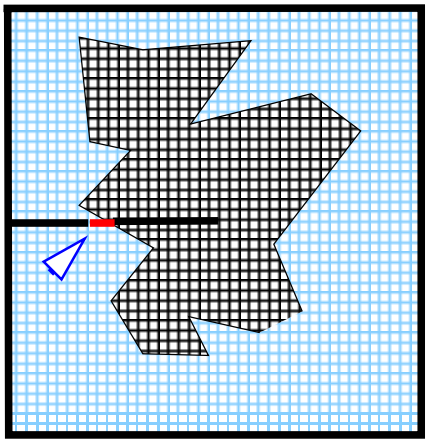


Figure 11.25: Method used to generate the droplets. After calculating the GS, several hard bonds (thick lines) are introduced, one hard bond is inverted (see triangle), leading to the appearance of an excitation (dark inner area).

Now we describe in detail which hard bonds are introduced to obtain droplets. First, all boundary spins are fixed relative to each other by introducing hard bonds around the border, see Fig. 11.25. The signs of these hard bonds are chosen such that they are compatible with the GS orientations of the spins they connect. This keeps the spins on the boundary in their GS orientations. Second, a line of hard bonds is created which runs from the middle of (say) the left border to a pre-chosen center spin, again fixing the bond's spins in their relative GS orientations. Next, the sign of exactly *one* hard bond on this line is inverted. Finally, a GS of the modified realization is calculated. With respect to the original GS, the result is a minimum energy excitation, fulfilling the constraints that it contains the center spin, does not run beyond the boundary and that it has a surface which runs through the hard bond which has been inverted. Hence, it will be a constrained droplet. The energy of this droplet is the energy of the resulting configuration calculated using the original bond configuration (i. e., without hard bonds).

Note that this procedure alone does not generate the droplets as defined above, because the border of the droplet is constrained to run through the inverted hard bond, while it can fluctuate freely for the original droplets. Therefore, for each realization, this procedure is iterated over all the bonds which are located on the line from the boundary to the center, when in each case exactly one hard bond is inverted. Furthermore, the full procedure is iterated over all four choices of lines of bonds running from the left, right, top and bottom boundary to the center spin. Among all $2L-2$ excitations generated in this way, the one exhibiting the lowest energy is selected as the final minimum energy droplet. This generates droplets very similar to the droplets of Aoki and Kawashima, except that no droplets can be generated, where the boundary fluctuates freely in all four directions at the same time. However, it was found that

these droplets play a minor role [168], their influence on the final result is smaller than the fluctuations resulting from the statistics.

Using this approach, droplets in systems up to size $L = 160$ could be studied [168]. This is smaller than the range of sizes studied for the domain walls, because here many, i. e., $2L - 2$, GS calculations are necessary to obtain just one droplet. For small sizes, the result was exactly the same as that found by Kawashima and Aoki. But at intermediate sizes a crossover occurs, and for larger sizes a power-law behavior is found, which is indeed compatible with $\theta' = \theta \approx -0.28$. This means, by going to larger system sizes, one central proposition of the droplet scaling theory could be verified for two-dimensional spin glasses.

Bibliography

[1] S. Kirckpatrick, C. D. Gelatt, and M. P. Vecchi, Science **4598**, 671 (1983).

[2] A. K. Hartmann and H. Rieger, *Optimization Algorithms in Physics*, (Wiley-VCH, Berlin, 2001).

[3] A. K. Hartmann and H. Rieger, *New Optimization Algorithms in Physics*, (Wiley-VCH, Berlin, 2004).

[4] H. Rieger, *Ground state properties of frustrated systems*, in: J. Kertesz and I. Kondor (eds.), *Advances in Computer Simulation*, Lecture Notes in Physics **501**, (Springer, Heidelberg, 1998).

[5] M. Alava, P. Duxbury, C. Moukarzel, and H. Rieger, *Combinatorial optimization and disordered systems*, in: C. Domb and J. L. Lebowitz (eds.), *Phase Transition and Critical Phenomena*, **18**, 141 (Academic Press, Cambridge 2000).

[6] M. E. J. Newman and G. T. Barkema, *Monte Carlo Methods in Statistical Physics* (Clarendon Press, Oxford, 1999).

[7] D. P. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, (Cambridge University Press, Cambridge 2000).

[8] P. J. M. Van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications (Mathematics and Its Applications)*, (Kluwer Academic Publishers, Dordrecht 1987).

[9] R. V. V. Vidal, *Applied Simulated Annealing* (Lecture Notes in Economics and Mathematical Systems **396**), (Springer, Heidelberg 1993).

[10] E. Marinari and G. Parisi, Europhys. Lett. **19**, 451 (1992).

[11] K. Hukushima and K. Nemoto, J. Phys. Soc. Jpn. **65**, 1604 (1996).

[12] U. Wolff, Phys. Rev. Lett. **62**, 361 (1989).

[13] H. G. Evertz, Adv. Phys. **52**, 1 (2003).

[14] C. Dress and W. Krauth, J. Phys. A **28**, L597 (1995).

[15] W. Krauth, in: A. K. Hartmann and H. Rieger (eds.), *New Optimization Algorithms in Physics*, 7 (Wiley-VCH, Berlin, 2004).

[16] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, J. Chem. Phys. **21**, 1087 (1953).

[17] M. E. J. Newmann and G. T. Barkema, Phys. Rev. E **53**, 393 (1996).

[18] D. A. Kessler and M. Bretz, Phys. Rev. B **41**, 4778 (1990).

[19] S. Liang, Phys. Rev. Lett. **69**, 2145 (1992).

[20] L. Santen and W. Krauth, Nature **405**, 550 (2000).

[21] M. N. Rosenbluth and A. W. Rosenbluth, J. Chem. Phys. **23**, 356 (1955).

[22] F. Wang and D. P. Landau, Phys. Rev. Lett. **86**, 2050 (2001).

[23] G. M. Torrie and J. P. Valleau, J. Comp. Phys. **23**, 187 (1977).

[24] B. Berg and T. Neuhaus, Phys. Lett. B **267**, 249 (1991); Phys. Rev. Lett. **68**, 9 (1992).

[25] S. Alder, S. Trebst, A. K. Hartmann, and M. Troyer, J. Stat. Mech. P07008 (2004).

[26] P. Grassberger and H. Frauenkron, in: P. Grassberger, G. T. Barkema, and W. Nadler. (eds.), *Workshop on Monte Carlo Approach to Biopolymers and Protein Folding*, (World Scientific, Singapure 1998).

[27] K. A. Dill, Biochemistry **24**, 1501 (1985).

[28] K. F. Lau and K. A. Dill, Macromolecules **22**, 3986 (1989); J. Chem. Phys. **95**, 3775 (1991); D. Shortle, H. S. Chan, and K. A. Dill, Protein Sci. **1**, 201 (1992).

[29] H. Frauenkron, U. Bastolla, E. Gerstner, P. Grassberger, and W. Nadler, Phys. Rev. Lett. **80** 3149 (1998).

[30] U. Bastolla, H. Frauenkron, E. Gerstner, P. Grassberger, and W. Nadler, Proteins: Struct., Fnct., Gen. **32**, 52 (1998).

[31] G. Zaránd, F. Pázmándi, K. F. Pál, and G. T. Zimányi, Phys. Rev. Lett. **89**, 150201 (2002).

[32] K. F. Pál, in: A. K. Hartmann and H. Rieger (eds.), *New Optimization Algorithms in Physics*, 205 (Wiley-VCH, Berlin, 2004).

[33] K. F. Pál, Physica A **233**, 60 (1996).

[34] M. J. Alava *et al.*, submitted to Phys. Rev. B, preprint cond-mat/0407297 (2004).

[35] K. F. Pál, Physica A **223**, 283 (1996).

[36] J. Houdayer and O. C. Martin, Phys. Rev. Lett. **83**, 1030 (1999).

[37] J. Houdayer and O. C. Martin, Phys. Rev. E **64**, 056704 (2001).

[38] M. Mitchell, *An Introduction to Genetic Algorithms*, (MIT Press, Cambridge (USA) 1996).

[39] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, (Springer, Heidelberg 1994).

[40] P. Sutton and S. Boyden, Am. J. Phys. **62**, 549 (1994).

[41] J. H. Holland, *Adaption in Natural and Artificial Systems*, (University of Michigan Press, Ann Arbor 1975).

[42] A. Yamaguchi and A. Sugamoto, Nucl. Phys. B Proc. Suppl. **83-84**, 837 (2000).

[43] I. Grigorenko and M. E. Garcia, Physica A **284**, 131 (2000).

[44] I. Grigorenko and M. E. Garcia, Physica A **291**, 439 (2001).

[45] A. Ulyanenkov, K. Omote, and J. Harada, Physica B. **283**, 237 (2000).

[46] T. X. Li, S. Y. Yin, Y. L. Ji, B. L. Wang, C. H. Wang, and J. J. Zhao, Phys. Lett. A **267**, 403 (2000).

[47] M. Iwamatsu, J. Chem. Phys. **112**, 10976 (2000).

[48] D. Romero, C. Barron, and S. Gomez, Comp. Phys. Comm. **123**, 87 (1999).

[49] Cheng Cheng, J. Phys. D **33**, 1169 (2000).

[50] R. S. Judson and H. Rabitz, Phys. Rev. Lett. **68**, 1500 (1992).

[51] F. G. Omenetto, B. P. Luce, and A. J. Taylor, J. Opt. Soc. Am. B **16**, 2005 (1999).

[52] D. G. Ireland, J. Phys. G **26**, 157 (2000).

[53] B. Ahrens, Meteor. Atmos. Phys. **70**, 227 (1999).

[54] H. Sadeghi, S. Suzuki, and H. Takenaka, Phys. Earth Plan. Inter. **113**, 355 (1999).

[55] M. Wahde, Astron. Astroph. Supplement Series **132**, 417 (1998).

[56] Genetic and Evolutionary Algorithm Toolbox, can be tested three weeks for free, see `http://www.geatbx.com/index.html`

[57] Matlab 5.3.1 is a commercial program for data analysis, numerical computations, plotting and simulation, see `http://www.mathworks.com/`

[58] S. M. Brown, *Bioinformatics*, (Eaton Publishing, Natick (MA) 2000).

[59] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis*, (Cambridge University Press, Cambridge 1998).

[60] A. K. Hartmann, in: A. K. Hartmann and H. Rieger (eds.), *New Optimization Algorithms in Physics*, 253 (Wiley-VCH, Berlin, 2004).

[61] T. Halpin-Healy and Y.-C. Zhang, Phys. Rep. **254**, 215 (1995) and references therein

[62] M. Kardar, G. Parisi, and Y.-C. Zhang, Phys. Rev. Lett. **56**, 889 (1986).

[63] M. Kardar and Y.-C. Zhang, Phys. Rev. Lett. **58**, 2087 (1987); T. Nattermann and R. Lipowski, Phys. Rev. Lett. **61**, 2508 (1988); J. Derrida and H. Spohn, J. Stat. Phys. **51**, 817 (1988); G. Parisi, J. de Phys. **51**, 1695 (1990); M. Mézard, J. de Phys. **51**, 1831 (1990); D. Fisher and D. Huse, Phys. Rev. B **43**, 10728 (1991).

[64] M. Kardar, Phys. Rev. Lett. **55**, 2235 (1985); D. Huse and C. L. Henley, Phys. Rev. Lett. **54**, 2708 (1985); M. Kardar, Phys. Rev. Lett. **55**, 2923 (1985).

[65] L. E. Reichl, *A Modern Course in Statistical Physics*, (John Wiley & Sons, New York 1998).

[66] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, Reading (MA) 1974).

[67] R. Sedgewick, *Algorithms in C*, (Addison-Wesley, Reading (MA) 1990).

[68] T. H. Cormen, S. Clifford, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, (MIT Press, Cambridge (USA) 2001).

[69] Y. Imry and S.-K. Ma, Phys. Rev. Lett. **35**, 1399 (1975).

[70] D. P. Belanger and A. P. Young, J. Magn. and Magn. Mat. **100**, 272 (1991).

[71] H. Rieger, in: D. Stauffer (ed.): Annual Reviews of Computational Physics **II**, 295, (World Scientific, Singapore 1995).

[72] A. P. Young (ed.), *Spin Glasses and Random Fields*, (World Scientific, Singapore 1998).

[73] K. Binder and A. P. Young, Rev. Mod. Phys. **58**, 801 (1986).

[74] M. Mezard, G. Parisi, and M. A. Virasoro, *Spin Glass Theory and Beyond*, (World Scientific, Singapore 1987).

[75] K. H. Fisher and J. A. Hertz, *Spin Glasses*, (Cambridge University Press, Cambridge 1991).

[76] J. A. Mydosh, *Spin Glasses: an Experimental Introduction*, (Taylor and Francis, London 1993).

[77] S. Fishman and A. Aharony, J. Phys. C **12**, L729 (1979).

[78] J. L. Cardy, Phys. Rev. B **29**, 505 (1984).

[79] D. P. Belanger, A. R. King, V. Jaccarino and J. L. Cardy, Phys. Rev. B **28**, 2522 (1983).

[80] D. P. Belanger and Z. Slanič, J. Magn. and Magn. Mat. **186**, 65 (1998).

[81] D. P. Belanger, in *Spin Glasses and Random Fields*, A. P. Young (Ed.), 251 (World Scientific, 1998).

[82] U. Nowak and K. D. Usadel, Phys. Rev. B **39**, 2516 (1989).

[83] U. Nowak and K. D. Usadel, Phys. Rev. B **44**, 7426 (1991).

[84] K. D. Usadel and U. Nowak, JMMM **104–107**, 179 (1992).

[85] T. Nattermann, in *Spin Glasses and Random Fields*, A. P. Young (Ed.), 277 (World Scientific, 1998).

[86] A. Aharony, Phys. Rev B **18**, 3318 (1978).

[87] M. Aizenman and J. Wehr, Phys. Rev. Lett. **62**, 2503 (1989).

[88] J. Bricomont and A. Kupiainen, Phys. Rev. Lett. **59**, 1829 (1987).

[89] T. Schneider and E. Pytte, Phys. Rev. B **15**, 1519 (1977).

[90] H. Rieger, Phys. Rev B **52**, 6659 (1995).

[91] M. E. J. Newman and G. T. Barkema, Phys. Rev. B **53**, 393 (1996).

[92] A. J. Bray and M. A. Moore, J. Phys. C **18** L927 (1985).

[93] D. S. Fisher, Phys. Rev. Lett. **56**, 416 (1986).

[94] J.-C. Picard and H. D. Ratliff, Networks **5**, 357 (1975).

[95] A. T. Ogielski, Phys. Rev. Lett. **57**, 1251 (1986).

[96] J. C. Anglès d'Auriac, M. Preissmann and A. Seb, J. Math. and Comp. Model. **26**, 1 (1997).

[97] J.-C. Picard and M. Queyranne, Math. Prog. Study **13**, 8 (1980).

[98] A. K. Hartmann and K. D. Usadel, Physica A **214**, 141 (1995).

[99] A. K. Hartmann, Physica A **248**, 1 (1998).

[100] S. Bastea and P. M. Duxbury, Phys. Rev. E **58**, 4261 (1998).

[101] E. T. Seppälä, M. J. Alava, and P. M. Duxbury, Phys. Rev. E **63**, 036126 (2001).

[102] E. T. Seppälä and M. J. Alava, Phys. Rev. Lett. **84**, 3982 (2000).

[103] E. T. Seppälä, V. I. Raisanen, and M. J. Alava, Phys. Rev. E **61**, 6312 (2000).

[104] L. R. Ford and D. R. Fulkerson, Canadian J. Math. **8**, 399 (1956).

[105] J. D. Claiborne, *Mathematical Preliminaries for Computer Networking*, (John Wiley & Sons, New York 1990).

[106] W. Knödel, *Graphentheoretische Methoden und ihre Anwendung*, (Springer, Berlin 1969).

[107] M. N. S. Swamy and K. Thulasiraman, *Graphs, Networks and Algorithms*, (John Wiley & Sons, New York 1991).

[108] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, (Princeton University Press, Princeton 1962).

[109] J. Edmonds and R. M. Karp, J. ACM **19**, 248 (1972).

[110] E. A. Dinic, Soviet Math. Dokl. **11**, 1277 (1970).

[111] R. E. Tarjan, *Data Structures and Network Algorithms*, (Society for Industrial and Applied Mathematics, Philadelphia 1983).

[112] A. V. Goldberg and R. E. Tarjan, J. ACM **35**, 921 (1988).

[113] B. Cherkassky and A. Goldberg, Algorithmica **19**, 390 (1997).

[114] A. V. Goldberg and R. Satish, J. ACM **45**, 783 (1998).

[115] K. Mehlhorn and S. Näher, *The LEDA Platform of Combinatorial and Geometric Computing* (Cambridge University Press, Cambridge 1999); see also `http://www.algorithmic-solutions.de`

[116] J. Siek, L.-Q. Lee, A. Lumsdainesee, *The Boost Graph Library*, (Addison-Wesley, Reading (MA) 2001); see also `http://www.boost.org/libs/graph/doc/index.html`

[117] A. A. Middleton, Phys. Rev. Lett. **88**, 017202 (2002).

[118] A. K. Hartmann and U. Nowak, Eur. Phys. J. B **7**, 105 (1999).

[119] J.-C. Anglès d'Auriac and N. Sourlas, Europhys. Lett. **39**, 473 (1997).

[120] M. R. Swift, A. J. Bray, A. Maritan, M. Cieplak, and J. R. Banavar, Europhys. Lett. **38**, 273 (1997).

[121] N. Sourlas, Comp. Phys. Comm. **121–122**, 183 (1999).

[122] A. A. Middleton and D. S. Fisher, Phys. Rev. B **65**, 134411 (2002).

[123] A. K. Hartmann and A. P. Young, Phys. Rev. B **64**, 214419 (2001).

[124] R. B. Potts, Proc. Camb. Phil. Soc. **48**, 106 (1952).

[125] F. Y. Wu, Rev. Mod. Phys. **54**, 235 (1982).

[126] R. J. Baxter, J. Phys. C **6**, L445 (1973).

[127] C. M. Fortuin and P. W. Kasteleyn, Physica **57**, 536 (1972).

[128] R. Juhász, H. Rieger, and F. Iglói, Phys. Rev. E **64**, 056122 (2001).

[129] A. Schrijver, *Combinatorial Optimization – Polyhedra and Efficiency Volume B*, (Springer, Berlin 2003).

[130] M. Grötschel, L. Lovász, A. Schrijver, Combinatorica **1**, 169 (1981).

[131] A. Schrijver, Journal of Combinatorial Theory Ser. B **80**, 346 (2000).

[132] S. Iwata, L. Fleischer, and S. Fujishige, Journal of the ACM **48**, 761 (2001).

[133] J.-Ch. Anglès d'Auriac, F. Iglói, M. Preissmann, and A. Sebő, J. Phys. A **35**, 6973 (2002).

[134] J.-Ch. Anglès d'Auriac, in: A. K. Hartmann and H. Rieger, *New Optimization Algorithms in Physics*, 101 (Wiley-VCH, Berlin, 2004).

[135] F. Barahona, J. Phys. A **15**, 3241 (1982).

[136] S. F. Edwards and P. W. Anderson, J. de Phys. **5**, 965 (1975).

[137] B. Müller and J. Reinhardt, *Neural Networks – An Introduction*, (Springer, Berlin Heidelberg, 1991).

[138] G. A. Kohring, J. de Phys. I **6**, 301 (1996).

[139] R. Floriana and S. Galam, Eur. Phys. J. B **16**, 189 (2000).

[140] M. Pasquini and M. Serva, Phys. Rev. E **63**, 056109 (2001).

[141] J. Maskawa, Physica A **311**, 563 (2002).

[142] G. Toulouse, *Commun. Phys.* **2**, 115 (1977).

[143] H. Rieger, L. Santen, U. Blasum, M. Diehl, M. Jünger, and G. Rinaldi, J. Phys. A **29**, 3939 (1996).

[144] N. Kawashima and H. Rieger, Europhys. Lett. **39**, 85 (1997).

[145] A. K. Hartmann and A. P. Young, Phys. Rev B **64**, 180404 (2001).

[146] J. Houdayer Eur. Phys. J. B **22**, 479 (2001).

[147] A. C. Carter, A. J. Bray, and M. A. Moore, Phys. Rev. Lett. 88, 077201 (2002).

[148] I. Bieche, R. Maynard, R. Rammal, and J. P. Uhry, J. Phys. A **13**, 2553 (1980).

[149] U. Derigs and A. Metz, Math. Prog. **50**, 113 (1991).

[150] F. Barahona, R. Maynard, R. Rammal, and J. P. Uhry, J. Phys. A **15**, 673 (1982).

[151] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, (John Wiley & Sons, New York 1998).

[152] B. Korte and J. Vygen, *Combinatorial Optimization – Theory and Algorithms*, (Springer, Heidelberg 2000).

[153] M. Palassini and A. P. Young, Phys. Rev. Lett. **85**, 3017 (2000).

[154] A. K. Hartmann and A. P. Young, Phys. Rev. B **66**, 094419 (2002).

[155] A. K. Hartmann and M. A. Moore, Phys. Rev. B **69**, 104409 (2004).

[156] N. Goldenfeld, *Lectures on Phase Transitions and the Renormalization Group*, (Addison Wesley, Reading (MA) 1992).

[157] J. Cardy, *Scaling and Renormalization in Statistical Physics*, (Cambridge University Press, Cambridge 1996).

[158] A. J. Bray and M. A. Moore, J. Phys. C **17**, L463 (1984).

[159] W. L. McMillan, J. Phys. C **17**, 3179 (1984).

[160] A. K. Hartmann, A. J. Bray, A. C. Carter, M. A. Moore, and A. P. Young, Phys. Rev. B **66**, 224401 (2002).

[161] W. L. McMillan, Phys. Rev. B **30**, 476 (1984).

[162] A. K. Hartmann, Phys. Rev. E **59**, 84 (1999).

[163] A. K. Hartmann, Phys. Rev. E **60**, 5135 (1999).

[164] K. Hukushima, preprint cond-mat/9903391 (1999).

[165] A. J. Bray and M. A. Moore, in J. L. van Hemmen and I. Morgenstern (eds.), *Glassy Dynamics and Optimization*, (Springer, Berlin 1986).

[166] D. S. Fisher and D. A. Huse, Phys. Rev. Lett. **56**, 1601 (1986); Phys. Rev. B **38**, 386 (1988).

[167] N. Kawashima and T. Aoki, J. Phys. Soc. Jpn., **69**, Suppl. A, 169 (2000). See also N. Kawashima, preprint cond-mat/9910366 (1999).

[168] A. K. Hartmann and M. A. Moore, Phys. Rev. Lett. **90**, 127201 (2003).

Index