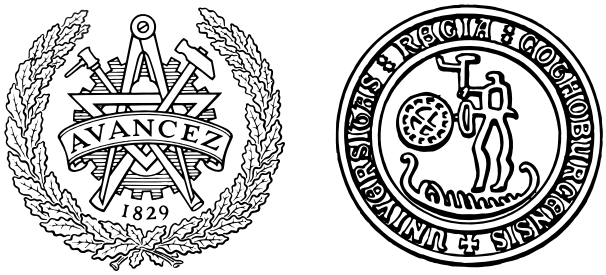
Relaxation in constraint satisfaction problems

Pontus Svenson

Licenciate Thesis



Institute for Theoretical Physics
Chalmers University of Technology
and Göteborg University

Göteborg 1999

Relaxation in constraint satisfaction problems

Pontus Svenson
tfkps@fy.chalmers.se

Institute for Theoretical Physics
Chalmers University of Technology and
Göteborg University
S-412 96 Göteborg, Sweden

Abstract

This thesis deals with some aspects of the physics of disordered systems. The thesis consists of two papers and an introductory part. The introduction briefly describes the theory of coarsening for regular lattice models and then gives an introduction to computational complexity theory. The definition of the Turing model of computation and of some important complexity classes are given, the Church-Turing hypothesis described, and the proofs of some important theorems reviewed. Paper I studies the relaxation behaviour using Monte Carlo simulation of some optimisation problems that, unless a very plausible conjecture in computer science is false, have worst case instances that require exponential time to solve. These problems can also be interpreted as spin glass models, and have previously been found to exhibit threshold phenomena akin to those of physical models undergoing phase transitions. In paper II, some preliminary results from Monte Carlo studies of a disordered ferromagnetic spin system are presented.

Key-words: Spin glasses, relaxation, computational complexity, NP-completeness, boolean satisfiability, graph colouring.

This licenciate thesis is based on the following papers, reprinted in the second part of the thesis and referred to as paper I and II in the introduction:

1. Pontus Svenson and Mats G. Nordahl, "Relaxation in graph coloring and satisfiability problems" , preprint `cond-mat/9810144` (to appear in Phys. Rev. **E 59** No. 4).
2. Pontus Svenson, "Monte Carlo studies of ferromagnetic Ising models on random graphs", preprint Göteborg ITP 99-1.

"Look, buddy, you didn't get polio!"
— Bob Laughlin, on the usefulness of science

Contents

Acknowledgements

I would like to thank past and present colleagues at the Institute for making it a nice place to work at, and also the journal club people for providing entertaining diversions during my doldrums. I thank Mats Nordahl for ideas and suggestions. Lastly, I am grateful to Neil Gaiman for inspiration and for lending me some words.

1 Introduction

There has always been strong ties between physics and computer science. Many of the pioneers in computer science also made significant contributions to physics, like von Neumann and Turing. The advance of solid state physics has made it possible to make computers cheaper and more powerful year after year. Physicist use these computers to perform ever more ambitious simulation studies of large systems of interacting particles and also of systems in non-traditional areas like artificial life [43] or realistic traffic models of real metropolitan areas [65]. The study of such complex systems is one of the areas in physics that has benefited most from modern computers, since large scale simulations are a necessity [70] in order to be able to obtain results in these areas.

Computer science has also benefited from physics. One example of this is the invention of new, physics-inspired algorithms to solve optimisation problems, like simulated annealing [36] or neural nets (e.g., [31]). Very recently, there has also been attempts to base search algorithms on more modern ideas from physics, such as renormalisation [35] and self-organised criticality [13]. Computer scientists studying machine learning have borrowed both models and methods to study them from statistical physics (e.g., [77]). More recently, this cooperation has been taken a step further with the discovery of phase transition phenomena similar to those in magnetic systems in optimisation problems [16, 32, 39]; see chapter 4. The subject of this thesis is the study of such optimisation (or constraint satisfaction) problems. These problems can also be interpreted as spin glass models, and are prototypical complex systems.

There are several possible definitions of the term "complex system". One viewpoint is to consider a system complex if it is difficult to solve. The $1D$ Ising model is unarguably less complex than the $2D$, which is in turn less complex than the (unsolved) $3D$ version. A more quantitative definition is to determine the complexity of the patterns that the model generates and let this be the complexity of the model. For example, the Ising model below the ordering temperature T_c is spatially homogeneous, without any complex patterns. Well above T_c there is no order and the system is completely random. But close to or at T_c, something different happens. There is no ordering, yet the patterns are not random. Coarse-graining (magnifying) the system results in a new system that has the same appearance as the old — the system possesses self-similarity. Patterns showing self-similarity are more complex than the others, and it takes longer time to simulate them on a computer.

It is important not to confuse complexity with randomness. Both completely ordered and completely random system should have a low complexity. Crutchfield and Feldman [20] have analysed the complexity of one dimensional spin systems using a complexity measure related to the amount of memory needed to predict the state of the system given knowledge of part of it; in [22] they review several different ways of measuring the complexity of patterns appearing in physical systems.

The computer science view of complexity is that a model is complex if it is difficult to simulate it on a computer or other machine. It is this aspect of complexity that motivates the

emphasis on computational complexity in this thesis, but it is important to bear in mind that this is not the complete definition of complexity or of complex systems. More discussion of these definitions can be found in [19]. In particular, the contribution by Anderson presents some arguments for why it is wrong to rely too much on the computer science definition of complexity in studying real world complex systems, and instead advocates for spin glasses as the proper paradigm for complex systems science.

A spin glass (e.g., [11, 53, 78]) is a system that has disorder and frustration. A simple example is the Edwards-Anderson model, which consists of spins in a d dimensional lattice with Hamiltonian $H = \sum_{ij} J_{ij} s_i s_j$, where the J_{ij} are random variables. The Sherrington-Kirkpatrick model is the case where the J_{ij} are Gaußian distributed, while the $\pm J$ model is the case where J_{ij} is $\pm J$ with equal probability if i and j are nearest neighbours and 0 otherwise. Experimentally realisable spin glasses include $Cd_{0.5}Mn_{0.5}Te$ and $Fe_{0.5}Mn_{0.5}TiO_3$; spin glasses are also relevant in the study of the normal phases of high-T_c superconductors. Disorder means that there is some amount of randomness in the interactions in the system, while frustration means that not all constraints can be satisfied at the same time. A model with ferro and antiferromagnetic interactions is frustrated if there is a loop such that the product of the interactions along it is -1. This can be interpreted as meaning that each spin in the loop will have an effective antiferromagnetic interaction with itself. Experimentally, the most important features of spin glasses are hysteresis and ageing effects.

Spin glass models and concepts have been used to study many different models in economics, biology, and social science and have also been used by several authors to study the physics of optimisation problems (e.g., [52, 7, 6, 32]). Spin glasses are important because it is believed that they capture many universal features of realistic models in most areas of complex systems, yet still are simple enough to study successfully.

This thesis is organised as follows. Chapter 2 contains a brief introduction to computer simulation using the Monte Carlo method and to phase ordering in simple models. Chapter 3 gives an introduction to computational complexity theory concentrating on **NP** and **NP**-complete problems. Some of the similarities between combinatorial optimisation problems and physical systems are described in chapter 4, while the concluding chapter 5 very briefly describes the included papers.

2 Computer simulation and phase ordering

How are computers actually used to obtain information on the physics of some system? In principle, this is trivial. For a spin system, the energy is calculated by averaging over all possible configurations of the spins, weighted by the Boltzmann factor $e^{-\beta E}$, where $\beta = \frac{1}{k_B T}$ is the inverse temperature and k_B is Boltzmann's constant. Other interesting quantities are determined by similar weighted averages. The problem with this is that if there are N spins than can each take on two different values, there are 2^N different configurations to sum over. This imposes a severe limitation on the system sizes that can be studied.

There is a much simpler method: Monte Carlo simulation [51]. This method determines the energy by a simple average over just a few of the possible configurations, without taking the Boltzmann factors into account.

Monte Carlo (MC) simulation works by starting in a random spin configuration S (i.e., one that is disordered, representative of the high temperature phase). The spin configuration is then changed into a new configuration S', most often by flipping a single spin (Glauber dynamics [28]) or exchanging two spins (Kawasaki dynamics [37]). The difference in energy ΔE of these two configurations is then calculated, and the new configuration accepted with probability $f(\Delta E)$, else rejected. There are various possibilities for f. In the work reported in this thesis we have used $f(\Delta E) = 1$ for $\Delta E \leq 0$, and $f(\Delta E) = e^{-\beta \Delta E}$ for $\Delta E > 0$. The Boltzmann factors are thus considered implicitly in the algorithm. Instead of multiplying the measurements with them and summing over all configurations, they determine which configurations the sum runs over. Normally, time is not increased until N new configurations have been tried; this is called one MC step per spin, or MC sweep, or epoch.

There are several other possible choices for f. The main limitation is that detailed balance must hold, i.e., the ratio between the probability to make a transition from configuration S to configuration S' and the probability of the inverse transition must be equal to the ratio of the configurations' Boltzmann factors (this is the reason for the choice of f above).

For $T = 0$, the Monte Carlo method reduces to a hill-climbing search procedure. It starts with a random configuration and tries to change this locally into a configuration with lower energy. This means that the MC method can get stuck in local minima. Finite temperature simulations also allow transitions that increase the energy, but with exponentially decreasing probabilities.

A variant of the MC method is the *simulated annealing* method [38]. Here the temperature is initially high but is reduced during the simulation. This minimises the risk of getting stuck in a local minima early on and improves the chance of reaching the global minimum searched for. Simulated annealing has been used to solve several optimisation problems with good results [36].

A general problem with the MC method is that care must be taken to ensure that the generated configurations are independent of each other. Data is not collected at each time step but instead with some granularity t_g. Normally, t_g is determined by requiring that some correlation function of the spin configurations is small. Another problem is that the initial

random configuration is often not characteristic of the temperature at which the simulation runs. The system must be equilibrated for a long time before measurements can start. The time needed for this equilibration diverges as T approaches the critical temperature as $\tau \sim \xi^z \sim |T - T_c|^{-\nu z}$, where ξ is the correlation length, ν the standard critical exponent relating the correlation length to the temperature, and z the dynamical critical exponent (see, e.g., [12]). This confirms that patterns get more complex (since it takes longer for simulations to determine them) as the temperature approaches T_c.

Some of the work presented in paper I and paper II studies the behaviour of the MC method during this equilibration. In paper I we study the $T = 0$ relaxation of the energy in a MC simulation starting from a random initial configuration for some hard optimisation problems. Paper II deals with the same subject for a ferromagnetic model on a random graph. In addition to being interesting in itself to study, this relaxation (or coarsening) behaviour also has applications in social science, were the goal is to study how an opinion or a rumour spreads through a population, and epidemiology, where it is important to know how a disease will spread.

Recently, a new measure has been introduced in the context of coarsening phenomena. Instead of just measuring the energy or the magnetisation of the spins in the model, it might also be interesting to look at how often a certain spin changes from up to down or vice versa. The fraction of spins that have not yet changed values is referred to as the fraction of persistent spins. This has direct applications in e.g., voter models, where it measures how often a given voter changes their opinion. A voter model is similar to a kinetic Ising model but has no Hamiltonian. Instead, each time a spin is considered a random neighbour is selected and the two spins are aligned with each other. This can be seen as a toy model for how opinions and rumours spread through a population. The most interesting property of voter models is that in dimensions 1 and 2 all the "voters" will eventually reach the same opinion, while in higher dimensions it is possible for minority opinions to survive.

Another interesting quantity to study is damage spreading. Here two or more system are simulated at the same time, using the same random numbers to determine configuration changes and whether or not they are accepted. The question asked is whether or not a small difference in the initial spin configurations spreads throughout the system or if it heals. Interestingly, the qualitative behaviour in damage spreading has been shown to depend on the updating order as well as the choice of algorithm [72, 73].

Figure 2.1 shows the behaviour of the energy

$$\epsilon = -\frac{1}{N} \sum_{<i,j>} s_i s_j \tag{2.1}$$

for the standard Ising model with N spins $s_i = \pm 1$ placed on a $2D$ square lattice using both Glauber and Kawasaki dynamics. The sum runs over nearest neighbours $< i, j >$ only. A power law relaxation is clearly seen. Note the anomalous behaviour for early times — this is an example of a cross-over behaviour, where there is one exponent for very early times and another for later times.

Experimentally, it is found that all models with scalar order parameter and discrete up down symmetry have the same behaviour (i.e, are in the same universality class) as the Ising model. In Halperin and Hohenberg's [34] classification, this is model A, while the universality class of the model with conserved order parameter is model B. Extensive simulation results for some models in these classes can be found in [67]. Note that there is also a hidden assumption that the interactions are local — random graph models do show different behaviour.

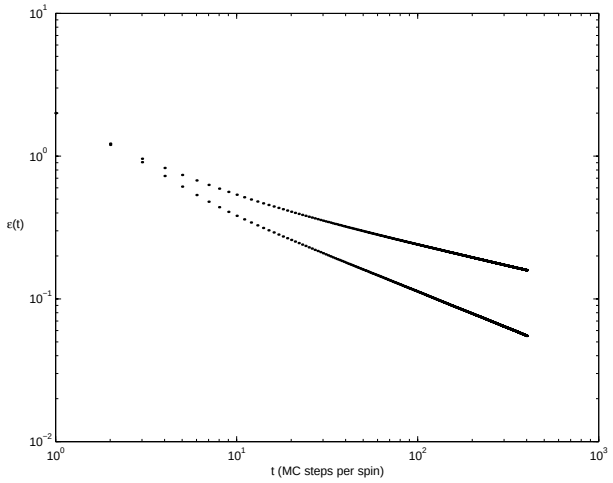


Figure 2.1: Energy relaxation from $T = 0$ Monte Carlo simulations of a square lattice Ising model with $N = 10^4$ spins averaged over 1000 different runs. Results are shown for both spin exchange dynamics, that conserve the order parameter, (upper curve) and single spin flip dynamics, which allow the magnetisation to change (lower curve). There is a clear power law relaxation $\epsilon \sim t^{-r}$ with $r = 1/3$ and $1/2$ for the different dynamics.

In order to explain the relaxation, we first need a continuum description of model A. The appropriate continuum description is the Ginzburg-Landau model, consisting of a order parameter field $m(\vec{r})$ and with free energy

$$F = \int d\vec{r} \frac{1}{2}(\nabla m)^2 + \frac{\mu}{2}m^2 + \frac{\lambda}{4}m^4 - hm \tag{2.2}$$

where h is the magnetic field. Arguments for the validity of this expression as the proper continuum version of the Ising model can be found in [12].

We are interested in how the energy behaves after a quench from a high temperature, disordered state into a temperature below the ordering temperature T_c. The system can not at once go into the composition that is favoured at this temperature. Instead small domains (droplets) that have the same value of the order parameter will form. If these droplets are sufficiently large, they will grow, otherwise shrink. When all droplets have disappeared and the system is homogeneous, the coarsening has stopped. Figure 2.2 shows these droplets in the $2D$ Ising model at $T = 0$ following a quench from a disordered state.

How can this growth of the droplets be described quantitatively? Recall equation 2.2; it gives rise to the following equation of motion for the order parameter

$$\frac{dm}{dt} = -\Gamma \frac{\delta F}{\delta m(\vec{r})} = -\Gamma[-\nabla^2 m + \mu m + \lambda m^3 - h] \tag{2.3}$$

where Γ defines the time scale of the system.

Assuming spherical symmetry and making an ansatz $m = m(r - R(t))$, this is rewritten as

$$-\frac{1}{\Gamma}\frac{dm}{dr}\frac{dR}{dt} = \frac{d^2m}{dr^2} + \frac{(d-1)}{r}\frac{dm}{dr} - \mu m - \lambda m^3 + h. \tag{2.4}$$

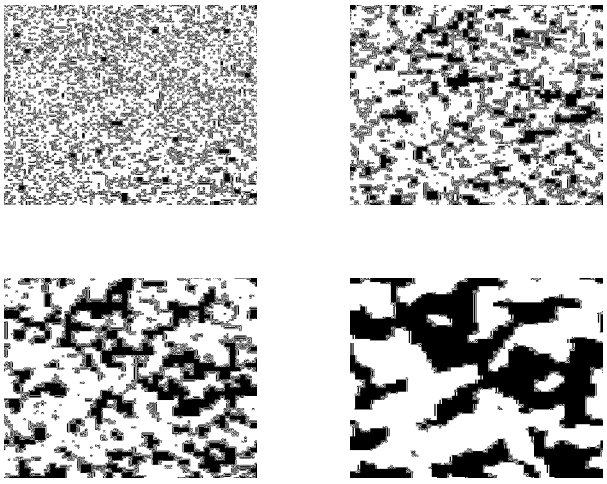


Figure 2.2: This figure shows the time evolution of the $2D$ square lattice Ising model at $T = 0$. Starting from a highly disordered state (top left), the system coarsens and droplets are formed.

We want to describe the domain wall, which is in the region $r \approx R$ and has a small width. It is thus possible to replace r by R in the equation. Introducing $v = \frac{dR}{dt}$ gives

$$\frac{d^2m}{dr^2} + [\frac{(d-1)}{R} + \frac{v}{\Gamma}]\frac{dm}{dr} - \mu m - \lambda m^3 + h = 0. \tag{2.5}$$

Equation (2.5) contains a dissipative term $[\frac{(d-1)}{R} + \frac{v}{\Gamma}]\frac{dm}{dr}$. In the absence of a magnetic field h, it can be argued [42] that this term must vanish, giving the velocity v of the interface as

$$v = \frac{dR}{dt} = -\frac{\Gamma(d-1)}{R(t)} \tag{2.6}$$

with solution

$$R(t) = \sqrt{R_0^2 - 2\Gamma(d-1)t}. \tag{2.7}$$

The interpretation of this result is that after a time t, only those domains whose length scale is larger than $R(t)$ will remain — the system will be ordered on all smaller length scales.

The length scale at which the system has ordered grows as a power law of time. What does this imply for the energy? A domain of radius R will have a surface energy contribution proportional to $\epsilon_d = R^{d-1}$ in d dimensions. If the total volume of the system is N, there will be on average $n_d = N/R^d$ such domains, leading to the total energy of the system being $\epsilon \sim \frac{1}{N}\epsilon_d n_d = R^{-1}$. From the result (2.7) for the growth of the length scale, we then get that the energy of the regular Ising model should grow as $\epsilon \sim t^{-1/2}$, which is consistent with the results from simulations shown in figure 2.1.

For the model with conserved order parameter things get more complicated, but it is possible to argue for the $\epsilon \sim t^{-1/3}$ law that is realised experimentally [42, 15]. This is a model where the ground state is not trivial to find. It can serve as a model for e.g., a system of two types of atoms that can diffuse but can not be converted into each other.

Other universality classes for coarsening phenomena have been identified by Lai, Mazenko and Valls [41]. In addition to model A and B, they introduce two types of system that have

logarithmic coarsening, $R(t) \sim R_0 + (AT \ln \frac{t}{\tau})^m$. Such logarithmic coarsening has been found in a simple tiling model by Shore et al [69].

For strongly disordered systems, additional complications arise, since there is currently no consensus on how the energy of an excitation relates to any length scale in the system, or even if there are length scales at all in the systems. For instance, figure 20 in paper I seems to indicate that for the graph colouring problem there are no domain structures in the normal sense.

Experimentally, coarsening can be measured using the equal-time structure factor

$$S(\vec{r} - \vec{r'}, t) = \langle m(\vec{r}, t) m(\vec{r'}, t) \rangle \tag{2.8}$$

which is independent of time in equilibrium, but shows a scaling behaviour $S(\vec{d}, t) = f(|\vec{d}|/R(t))$ during equilibration.

3 Computational complexity

3.1 Introduction — not everything can be computed

"Not knowing everything is all that makes it okay, sometimes."
— Delirium, in Brief Lives

Ultimately, computer science is about solving problems. For each problem there are several different algorithms than can be used to solve it. Which one of these algorithms is the "best" one to use? One way of determining this could be to use the one that is most beautiful in the mathematical sense, but for practical purposes it is more interesting to look at the amount of resources the algorithm needs to solve the problem. The most interesting resources are time and space. If we can only solve a problem by using 10^{23} bytes and 10^2 years, the problem is for all practical purposes unsolvable. Since all small problems are easy to solve, computer scientists interest themselves with how the amount of resources needed to solve a problem scales with its size, N. The size of a problem is a somewhat ambiguous quantity; it is defined as the amount of memory needed to represent an instance of it. For matrix multiplication, N could be either the total number of elements in the matrix or the number of rows or columns (whichever is largest). If we want to factorise a number x, N is the number of bits needed to represent it, i.e, $N = \log x$.

In addition to helping physicists simulate more and more complex systems, computers have also helped mathematicians in providing proofs for some problems that has so far eluded the search for simple proofs. The most famous example of a theorem that can (so far) only be proven with the help of a computer is the Four Colour Theorem, which states that four colours suffice to colour a two dimensional map of countries [8, 9]. Another example is the proof of the Robbins conjecture by an automated theorem prover in 1996 [50]. A boolean algebra is a set with an unary negation operator $\neg$ and and a binary operator $\vee$ fulfilling associativity and commutativity as well as $\neg(\neg x \vee y) \vee \neg(\neg x \vee \neg y) = x$. The Robbins conjecture states that this third axiom can be derived from the Robbins equation $\neg(\neg(x \vee y) \vee \neg(x \vee \neg y)) = x$. In contrast to the proof of the Four Colour Theorem, the proof of the Robbins conjecture is short enough that it is possible to check it by hand.

Computers can also be used to discover new knowledge. Perhaps the best example of this is the "Assistant Mathematician" program written by Lenat (e.g., [44]) in the 70's. This program, which started with the notion of sets and some simple rules to determine what is "interesting", managed to discover not only integers, addition, and multiplication, but also prime numbers and several elementary results from number theory, such as the Goldbach conjecture. Even more surprising is that it also discovered the concept of maximally divisible number (a number that has more divisors than any number smaller than it, e.g., 12) which, although studied by Ramanujan, is a concept still unknown to many mathematicians. (Note that the program did not prove anything, it only made conjectures regarding what appeared interesting to it.)

Despite these successes there are still many thing that computers are unable to do. One of the problems in modern artificial intelligence (e.g., [76]) is that even though it is possible to make systems that show a certain amount of intelligence in restricted domains it is still not possible to make a program that has the same general level of intelligence and common sense as a human.

For these reasons it is important to know what limits there are on the power of computers. We can distinguish two different types of questions. The first regards what can be done at all, the second what can be done with limited resources.

It is quite easy to demonstrate that there must exist functions that are uncomputable by computer: the number of programs that can be run on a specific computer is countably infinite, while the number of functions, taking a number as input and giving a number as output, is uncountably infinite.

One of the simplest physical problem that is unsolvable is the tiling problem. Here we are given a set of shapes and the goal is to tile the plane with them. This is of course a very simple problem if the number of distinct shapes in the set is restricted (e.g., if the set consists of just one polygon, we know what shapes it may be in order to be able to tile the plane). But the general problem of determining if an arbitrary set of shapes can tile the plane is uncomputable; a readable proof of this for a simple variation of the tiling problem can be found in [75].

Another example of an unsolvable problem is the halting problem. Here the objective is to determine if a computer program will halt or if it loops forever. Assume that there is a program $H(x, y)$ that determines if the program represented by x will halt on input y. Consider the following program M, assumed to take x as input:

```
if(H(x,x)) then
    loop forever;
else
    return 1;
```

It simply determines whether or not the program input to it will halt given itself as input. Now consider computing $M(M)$. If this computation halts, the conditional in M will ensure that it loops forever. But if it does not halt, $H(x, x)$ will return false, which means that the else branch will be taken, so that M halts. The only way to resolve the contradictions is to reject the existence of the machine that computes H, which proves the uncomputability of the halting problem.

The books by Feynman [23] and Dewdney [21] give good elementary introductions to the theory of computation.

3.2 Some problems are harder than others

Computer scientist distinguish between problems that are tractable and those that are intractable. Intractable problems are those for which it is known that the fastest way to solve them requires exponential time, while a problem is tractable (or feasible) if there is a polynomial time algorithm to solve it. The border line between these two types of problems is of course fuzzy, since there are a lot of problems for which it is unknown whether they can be solved in polynomial time or not.

Perhaps the simplest example of an intractable problem is the Towers of Hanoi problem. In this puzzle, we are given three bins. One of the bins has N plates on it, sorted in decreasing size from bottom and up. The goal is to move all the plates to another bin. The rules are that we are

Figure 3.1: The Towers of Hanoi puzzle involves moving the plates one at a time from the leftmost to the rightmost bin in such a way that there never is a plate with a larger plate above it.

only allowed to move one plate at a time and that all configurations where a larger plate resides on top of a smaller are forbidden. Figure 3.1 shows an instance with $N = 5$. This problem has beautiful recursive and iterative solutions, but it is easy to show that there is no way to solve it using less than exponential time. Hence, it is an intractable problem.

Some problems abruptly change character when changed just slightly. Consider the problem of dividing a set of numbers into two subsets of sizes n_1 and n_2 so that the difference between the sums of the numbers in set one and those in set two is maximised. This is a trivial problem — just place the largest numbers in one set and the smallest in the other. This can certainly be done in polynomial time. If, on the other hand, the goal instead is to minimise the difference, the problem gets much tougher — the only currently known solution is to try all possible partitions of the set, which takes exponential time.

Another example comes from graph theory. Consider the Euler problem — is there a path through the graph that starts and ends on the same node and visits each edge exactly once? A related problem is the Hamiltonian circuit problem (or travelling salesperson (**TSP**) problem): is there a path of length less than l such that each node is visited exactly once? The former problem is trivial to solve in polynomial time, while the latter is **NP**-complete (see below) and thus almost certainly has no algorithm that runs in polynomial time for all inputs.

Computer scientist have most often only bothered with worst-case analysis of algorithms, but the phase transition found in some constraint satisfaction problems further stresses the importance of obtaining more refined tools to analyse a problem instance's difficulty. It is similar to the situation with sorting lists — here the fastest known [40] algorithm has a worst-case complexity of $N \log N$. But the algorithm that is most widely used has a worst-case complexity of N^2, but an average-case complexity of $N \log N$ and with a smaller constant, and is hence the one that is used most often.

3.3 The Turing machine

In order to compare the complexities of algorithms with each other, it is important to use the same tools to analyse them. For this a universal model of computation is needed. One of the simplest such models is the Turing Machine.

A Turing machine consists of a tape and a control mechanism that can read from, write to, and move the tape. We must also specify an alphabet of symbols that may be written on the tape and a set of internal states that the Turing machine may be in. In each time step the control mechanism determines the next state by looking at the tape and its current state and then consulting a transition table that tells it what it should do for all possible combinations of current state and read symbol. The transition table tells the control mechanism what state the machine should enter next, which way it should move the tape, and what symbol it should write in the current cell of the tape (either overwriting the old symbol or just copying it).

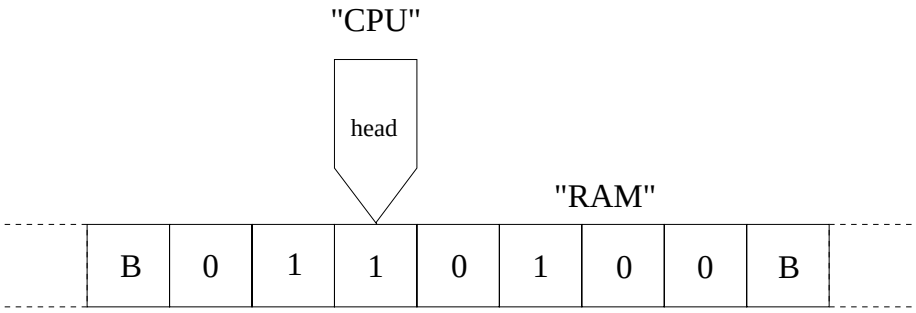


Figure 3.2: Schematic diagram of a Turing machine. It consists of a tape ("memory") and a tape control head ("CPU"). Each cell of the tape contains either a symbol from the alphabet of the machine (here 0 or 1) or is blank (denoted B here). In each time step, the head scans the current cell, changes its internal state and moves one step to the left or right, possibly changing the contents of the cell.

The definition of the Turing machine is roughly based on the way a human being computes things. If a person is given a complicated problem to solve, they will solve it step by step. Each step consists of some simple computation that can be done in the head, followed by writing down the result of this computation on paper. The state of the person's brain also changes in each step, to reflect the work that has been done and that which is left to do. For some steps, the person might need to make reference to previous results on the paper. This is completely analogous to a Turing machine — the paper is the tape, the human brain is the tape control unit. We do not yet know the full transition table of the human computer, but the subset used for some simple problems can be determined.

It is easy to se how to construct Turing machines that solve simple problems. But a modern computer can be programmed so that it can be used to solve not just one problem but any (computable) problem. In the same way it is possible to construct Turing machines that are universal in the sense that they can simulate other Turing machines, designed for solving specific problems. The way to do this is to allow the machine to have two inputs — in addition to the data on which it is to work, it is also provided with a program that contains a complete description of the Turing machine that it should simulate.

Universal Turing machines can be constructed that are very small: Rogozhin [66] has made one version with 4 states and 6 symbols and one with only 2 states and 18 symbols, while Minsky [54] has one with 7 states and 4 symbols. Since the universal Turing machine must simulate the dedicated Turing machine it is often much slower than the Turing machine it simulates. But if the problem takes time t to solve using a dedicated Turing machine, the time needed for a universal Turing machine to solve it is at most a polynomial of t — thus it is possible to simulate other machines efficiently.

There are various modifications that can be made to the Turing machine that do not change its computing capabilities. The tape can be either infinite or semi-infinite (i.e., it has a start but no end), or the machine could even have access to several tapes. None of these additions change anything substantial about the machine, and the more advanced versions can all be simulated (with a polynomial slowdown) by a single tape Turing machine. To simulate a machine with k tapes, the universal Turing machine simply divides its tape into k different parts, adding special markers to its alphabet to see where the tapes start and end. Special care must be taken when the tapes grow, but the details are trivial. The machines used in analysing algorithms often have

separate tapes for input and output, and universal Turing machines also often have a separate tape for the program they are running.

There are some examples of where Turing machines have been used in "proper" mathematics. Matiyasevich has used Turing machines to show that Hilbert's 10th Problem (to give an algorithm that solves a Diophantine equation) is unsolvable (e.g., [48]), and to produce a polynomial in 10 variables whose non-negative values give exactly the set of all prime numbers [47]. These results are not important from a practical point of view, since there are far better methods to test for primality, but they are important from a conceptual point of view of mathematics.

3.4 Complexity classes and problems

We will often talk about the time complexity (or just complexity) of a problem. It is understood that this actually means the time complexity of the best known algorithm that solves that problem.

It is convenient to group together all those problems that can be solved by a universal Turing machine with time complexity that is any polynomial in the problem size N. This class is called **P**. Sometimes it is enough to know with some probability that a problem has a solution. The class of problems for which it is possible to find a solution with probability 2/3 in polynomial time is called **BPP**(for bounded error probability polynomial). The choice of 2/3 is of course arbitrary, any probability larger than 1/2 would do just as well. The class **BPP** can be called the class of problems that can be solved efficiently. (The quantum analogue of this class is important for quantum computers. Some evidence that it is more powerful than its classical counterpart has been obtained in [10].)

Why are there no separate classes for problems that require time proportional to N, N^2, N^3, and so on? The reason is that this difference can be removed by considering more advanced version of the Turing machine. If the computer has superinstructions that are the equivalent of several smaller instructions, it is possible to show that the coefficient of all but the linear part of the polynomial can be made arbitrary small. This means that there is no point, from a theoretical computer scientist point of view, to argue about whether a problem can be solved in N or N^{30} time. (It is also a fact that most problems than can be solved in polynomial time have solutions that are linear, quadratic or at most cubic (matrix multiplication) in problem size.)

There are many important problems for which it is not known if a polynomial time algorithm exists. For some of these problems it is however possible to verify that a proposed solution actually is a solution in polynomial time. So if we guess a candidate solution, or if a friendly genie gives us a candidate, we can see in polynomial time if the guess was correct.

The class of all problems for which this it true is called **NP** (for non-deterministic polynomial). The name comes from the fact that a Turing machine that is allowed to make non-deterministic choices in its computation would be able to solve it in polynomial time.

The archetypical **NP** problem is boolean satisfiability. This is the problem of determining whether or not the variables in a given boolean formula can be assigned so that the formula evaluates to true. For example, $x \wedge y$ can be satisfied by setting both x and y to true, while $x \wedge \neg x$ obviously can not be satisfied. A special form of satisfiability is k-**SAT**, where the formula is written in conjunctive normal form, that is, it consists of the logical and of M clauses, each clause being the logical or of k possibly negated variables. For instance, $(x \vee y) \wedge (\neg x \vee z)$ is an instance of 2-**SAT** with two clauses and three variables. It is easy to see that a general boolean formula can always be written in conjunctive normal form.

It is interesting to note that **Horn-SAT**, a special instance of 3**-SAT** where each clause contains at most one unnegated variable, is in **P**. It can be solved in polynomial time using resolution (e.g., [76]). This method takes advantage of the fact that all Horn-clauses can be written as an implication, e.g., $\neg x \vee \neg y \vee z$ is equivalent to $x \wedge y \rightarrow z$. Also 2**-SAT** can be solved in polynomial time. This algorithm is based on the fact that a clause $c_1 \vee c_2$ (where c_i can be either a variable or the negation of a variable) means that if $\neg c_1$ is true, then c_2 must also be true. We introduce a directed graph whose nodes are the literals x and $\neg x$, where x is a variable in the formula, and interpret each clause as two edges in this graph, one between $\neg c_1$ and c_2 and the other between $\neg c_2$ and c_1. Satisfiability of the 2**-SAT** formula can be checked simply by seeing if there is a loop from x to $\neg x$ and back to x for some variable x. If there is such a loop, the formula is clearly not satisfiable.

All problems that are in **P** are of course also in **NP**, but it is currently an open question whether or not there are problems in **NP** that are not in **P**. The problems in **NP** most likely not to be solvable in polynomial time are the so called **NP**-complete problems. This class is related to relative difficulties of problems. If a problem π is such that an effective way to solve it would mean that all problems in some class **A** could also be solved efficiently, π is said to be **A**-complete.

The type of problems that are most often used in complexity theory are the decision problems. Here we are not given a function to minimise or maximise (as in physics or economy). Instead, the task is to determine if some specific input belongs to a given set, such as the set of all graphs that are colourable using at most three different colours. If is easy to see that an optimisation problem can always be converted into a decision problem. Instead of asking for the best solution we simply ask whether there is a solution whose fitness (defined in some way) is better than a bound x. An efficient algorithm for the optimisation version of the problem can thus be used to solve the decision problem efficiently.

It is perhaps less obvious that the converse is also true. But given an efficient method to determine if a solution that is better than x exists, better and better approximations to the best value can be obtained using binary search, i.e., by determining which interval the best solution lies in and then halving this repeatedly until the desired accuracy is attained. After the best value has been determined, the problem can be manipulated (by e.g., restricting the domains of some variables or changing their interactions) to determine the variable assignments in a solution. In the satisfiability problem, for example, the value of a variable x_l can be determined by considering a new formula where x_l is set to true. If this formula is satisfiable there is a solution with x_l true, otherwise there must be a solution with x_l false. In either case we can continue with the next variable and determine its value. Note that this procedure does not make it possible to determine all solutions to the problem, it only provides one of them. Enumerating all the solutions requires exponential time.

Another important concept is that of the complement of a problem. The complement of a decision problem π is simply the same problem but with opposite answers, i.e., if π' is the complement of π, then $x \in \pi'$ if and only if $x \notin \pi$. The complement of satisfiability is the problem of determining that there are no assignments that satisfy a given formula. The complement of a problem that is in **P** is also in **P**, i.e., **P** = **co-P**. This fails however for **NP**, since an evidence for a yes instance can not be converted to saying anything about whether or not the complemented problem has a solution. This is an important distinction between deterministic and non-deterministic computation.

To define completeness, we need to formalise the way a problem can be used to solve other problems. If a solution to π can be converted into a solution of π', we say that π' reduces to π.

There is some ambiguity in the definitions of a reduction in the literature. In most textbooks, the exact definition is glossed over and it is simply said that a reduction should require at most polynomial time. A more general definition is that a reduction is a conversion between two problems that requires space that is at most logarithmic in problem size. (It can be shown that this implies that it can not use more than polynomial time). Completeness can now be defined formally. A problem π is complete for a class **A** if π is in **A** and all other problems in **A** can be reduced to π. The class **A**-complete thus consists of the most important, most difficult problems in **A**. If we can solve a **NP**-complete problem efficiently, we have shown that **P** =**NP** since all other problems in **NP** can be reduced to π and hence solved in polynomial time. A problem is said to be **NP**-hard if all **NP** problems can be reduced to it but it is not necessarily in **NP**.

In addition to the time complexity classes introduced here, it is also possible to use space as a resource. For example, **PSPACE** is the class of all problems that can be solved by a universal Turing machine with no time limits but using at most a polynomial amount of memory. There are very interesting asymmetries between time and space. For instance, while it is open whether or not **P** = **NP**, it is known that **PSPACE** is the same as the class of all problems solvable in polynomial space by a non-determinstic Turing machine, so that the addition of non-determinacy does not have any influence on the memory requirements of computations.

NPC is the class of all problems that are in **NP** and are at least as hard as all other **NP** problems, in the worst-case sense. There has also been some work on the concept of *average case complexity*, leading to the introduction of the class **AvP**. **AvP** is the class of all problems such that the average time needed to solve them is bounded by a polynomial. The average is here taken with respect to some probability distribution of the inputs to the problem. See [18] and references therein for a more complete discussion of various forms of average complexity classes.

3.5 $\exists$ NPC

The first problem shown to be in **NPC** was the satisfiability problem. The proof can be found in detail in [61] and [46], here the main ideas will be sketched. **SAT** is in **NP** since it is trivial to check in polynomial time whether or not a given assignment of the variables in a formula really satisfies the formula.

Since we do not even know all problems that are in **NP**, it seems a formidable task to be able to show that all problems in **NP** reduce to **SAT**. The proof relies on the insight that if a problem π is in **NP** then there is a non-deterministic Turing machine that will solve it. We will introduce a way of describing the computation that this non-deterministic Turing machine must do, and then show that this description is in fact a boolean formula, where there are variables representing the complete configuration of the Turing machine at each time step, including the non-deterministic choices that the machine can make. Solving this boolean formula (i.e., the satisifiability problem) in polynomial time would then enable us to construct explicitly a Turing machine that will solve π in polynomial time. But this is the definition of completeness, and hence boolean satisfiability is **NP**-complete.

As argued for above, it is no restriction to assume that π is a decision problem, i.e., given some input the problem is to answer yes or no. We will also assume that there is some specific location on the tape that serves as the output of the program; the contents of this cell will then be taken to give the value of the formula. How can the operation of a non-deterministic Turing machine be described? In each time step, we have to keep track of which state the control mechanism is in and the contents of the tape. If the size of the input data is n, we know that

the machine will terminate after at most n^k steps, for some k. The configuration of the Turing machine at time t will corespond to the t'th row in a table with n^k rows.

How many columns are needed in the table? The tape is infinite, so it would seem that an infinite number of columns would be needed, but this is not the case. Since the tape head can move only one step at each time step it is enough to have n^k cells of tape. (This is a general feature of models of sequential computation: they can not consume more space resources than time.)

In order to describe also the current state of the machine, the alphabet of the machine will be modified subtly. The cell where the tape head is currently positioned will be marked with a symbol that encodes both the contents of the tape at that cell and the current state of the control mechanism.

We also need to capture the non-determinacy of the machine in some way. This is done by introducing choices c_i for each time step i. These choices can represent for instance coin-tosses in a randomised algorithm or the kind of oracle-queries allowed for a non-deterministic Turing machine. It is no restriction to assume that at each step there are two choices, so that $c_i = 0$ or 1.

Now it is easy to see how to convert the description of the machine into a boolean formula. First note that the top row of the table will be fixed by the input data (for technical reasons, it is also necessary to assume that the extreme right and left columns are fixed to be the blank symbol, see [61] or [46] for details). Given a row, we now need to determine how the row beneath it will look like. The only cells that may change are the one at which the tape head is currently positioned and its nearest neighbours (if the head moves to them). The new states of these cells are uniquely determined by the states of the cells above them and the cell determining the non-deterministic choice, and they can in fact be determined by a boolean circuit with those cells as input (we ignore trivial and uninteresting details such as encoding the alphabet using only 0's and 1's, and refer the interested reader to the references). This construction can be continued for all the rows of the table, and it is clear that the boolean formula that determines the state of the designated output cell after n^k steps can be constructed in polynomial time.

By solving this boolean formula, the operation of the Turing machine solving π can be determined, including the correct values of the non-deterministic guesses c_i. (Since π was assumed to be in **NP**, the formula is guaranteed to have at least one solution.)

There are today many more problems that have been shown to be **NP**-complete. To do this for a given problem π it suffices to show that **SAT** can be reduced to π — since all problems in **NP** can be reduced to **SAT**, it follows that any problem in **NP** can be reduced to π by composing the two reductions. The book by Garey and Johnson [26] has an extensive list of **NP**-complete problems; more can be found online at
`http://www.nada.kth.se/~viggo/problemlist/compendium.html`.

Another important **NP**-complete problem is graph colouring. This is the problem of colouring all the nodes in a graph so that no node is connected to another with the same colour. It is related to the Four Colouring Theorem, and has a very simple physical interpretation: it is the problem of determining a ground state with zero energy of an antiferromagnetic Potts model. The number of states q in the Potts model corresponds to the number of allowed colours in graph colouring. Graph colouring is also intimately related to all sorts of scheduling problems. Here each event is a node in a graph, and two nodes are linked if and only if the events they represent must not take place at the same time. The number of timeslots that are available corresponds to the number of colours allowed to colour the graph.

Many interesting results can be shown concerning graph colouring. For instance, it is known

that there exists graphs which are not colourable with three colours yet contain no triangles [14].

Another **NP**-complete graph problem is graph partitioning. Here the nodes of a graph are to be divided into k subsets so that no edges join two nodes from different subsets. Physically, this corresponds to a model with ferromagnetic interactions and where there is also some condition on the magnetisation. Many spin glasses have also been shown to be **NP**-complete, including the Sherrington-Kirkpatrick and $3D$ $\pm J$ models.

In general, we can define a constraint satisfaction problem (or **CSP**) as a problem with N variables x_i and M constraints c_i. In general the variable x_i could take on the values $0, \ldots, d_i - 1$, but often the simplification that all $d_i = d$ is made. Each of the constraints consists of a list of variables and the combinations of these variables that are not allowed. A constraint involving two variables can be seen as a matrix where the rows and columns correspond to the first and second variable, respectively. For graph colouring the constraints are diagonal matrices for each of the edges in the graph. The ratio between the number of constraints and the number of variables turns out to be an important parameter. In graph colouring, the connectivity or average degree of the graph, $\gamma = \frac{1}{2}\frac{M}{N}$ is used, while for k-**SAT** it is the ratio of the number of clauses and the number of variables $\alpha = \frac{M}{N}$ that is the relevant parameter.

Mitchell [55] considers a wide variety of algorithms to solve constraint satisfaction problems and shows that they are all guaranteed to take exponential time on infinitely many instances, hence lending strong support to the notion that **P**$\neq$ **NP**.

3.6 The structure of NP

There are many interesting questions about **NP**. As stated above, the **P** = **NP** question is perhaps the major open question in theoretical computer science. One of the most beautiful results is that if **NP** $\neq$ **P** then there must be problems in **NP** that are in neither **P** nor **NPC**; the class of all such problems is called **NPI** (for "**NP** intermediate").

The proof of this result consists of an explicit constructions of a problem π in **NPI**. The same method can then be used to construct a problem that is more difficult than all **P** problems but not as difficult as π. This process can be continued indefinitely, so that if **P** $\neq$ **NP**(which is almost certainly the case), there is actually an infinite hierarchy of problems between the "easy" ones in **P** and the "difficult" ones in **NPC**.

π will be constructed in such a way that if it can be decided in polynomial time, then it will be the same as satisfiability, which by assumption is not in **P**. Similarly, if π were in **NPC**, the construction of it will ensure that it is in fact a trivial problem, hence solvable in polynomial time, resulting in the same contraditcion. The only way to resolve these contradictions is if π is in **NPI**.

π will be very simple: given a string x that represents a boolean formula as input, x will solve π if and only if x is a valid and satisfiable formula, and a certain function $f(n)$ where n is the length of x is even. If we can construct $f(n)$ so that it is even if π is in **P** and odd if π is in **NPC**, π will have the required properties and the proof that **NPI** exists will be complete.

The construction of f will ensure that these conditions are met for all inputs of size larger than N_0, where N_0 is some constant. This is enough for the proof, since it is the time complexity as input size goes to infinity that determines the complexity class of π. A finite number of exceptions can always be handled by special cases.

The Turing machine that computes f will work by examining in turn all polynomial time Turing machines and all reductions, giving them all strings as input. We therefore need to assume that there is an enumeration of all polynomial time Turing machines M_i and of all reductions R_i

(i.e., R_i is the Turing machine that performs reduction i). We also need to equip the machines with a stop-watch, so that we can halt them after t time steps. Both of these assumptions are trivial; they are necessary because we need to try the M_i and R_i on progressively longer strings. If a M_i is found that seems to determine π, the output of f will be even. If, on the other hand, a reduction is found that seems to reduce **SAT** to π, f will be odd. This ensures that the promised contradictions arise. f will change parity (and hence move on to trying the next machine or reduction) as soon as it has determined that the Turing machine or reduction currently under test does not correspond to a machine that computes π or reduces **SAT** to it.

The computation of $f(n)$ will consist of 2 steps. Each step lasting for exactly n time steps. During the first step, the tape-head of the Turing machine will advance to the right on the tape, and it will also calculate $f(j)$ for as many $j = 1, 2, \ldots$ as it has time. Let the last such value calculated by $f(j_{max}) = k$. If k is even, let $i = \frac{k}{2}$, otherwise $i = \frac{k-1}{2}$.

The next step of the calculation will now determine the value of $f(n)$ — it will be either k or $k+1$. If k is even, we will try to use the Turing machine M_i, patiently run through all possible strings of length up to n as input data, and try to find some string for which this machine does not give the same result as the machine determining π. If such a string is found, then the computation terminates and $f(n) = k+1$, if such a string is not found before time has run out, we let $f(n) = k$.

One of the contradictions can now be seen easily. For if π is indeed in **P**, then there is some machine M_l that determines it. For all strings that are sufficiently long, f will thus be a constant function, since we can never find a string for which M_l and the machine computing π do not agree. Furthermore, the output of f will be an even number. But this means that for all sufficiently long strings, π is the same as **SAT**, which by assumption did not have a polynomial time solution. So something must be wrong, and we conclude that π can not lie in **P**.

It is now simple to see what must happen if $f(n)$ is odd. In an exactly analogous way, we now test all possible reductions on all possible strings, until we either find one string that can not be reduced to π by the currently tested reduction or time runs out. By similar reasoning as above, if there is a reduction of **SAT** to π (which there must be if π is to be **NP**-complete), then the function f will be constant and odd for all sufficiently long strings. But this means that π is, apart from a finite number of strings, a trivial problem and hence certainly in **P**. Again we have reached a contradiction, and we can conclude that if $\mathbf{P} \neq \mathbf{NP}$ then there are indeed problems that are harder than any **P** problem yet easier than any **NPC** problem.

3.7 Even harder problems

There are several other complexity classes that can be defined in similar way as **NP**. For instance, the subset of **NP**-problems whose optimisation versions do not ask for a bound for the fitness but ask if the fitness is exactly x is called **DP**. Formally, this is the class of decision problems l than can be written as the intersection $l_1 \cap l_2$ where $l_1 \in \mathbf{NP}$ and $l_2 \in$ **co-NP**. The problems l_1 and l_2 are the ones that ask if there is any solution to the problem with better or worse fitness than x, respectively. For more details on **DP**, see [61]. These problems are much harder than those in **NP**, since they require the ability to check both if a problem has a solution and also if another one does not have a solution. Note that **DP** is not the same as the intersection of **NP** and **co-NP**.

Another extension of **NP** is the possibility of adding an oracle to the Turing machine. Here the Turing machine is equipped with a black box that can instantly solve some problem π. We call the set of problems that this extended Turing machine can solve in polynomial time $\mathbf{P}^{\pi}$, while

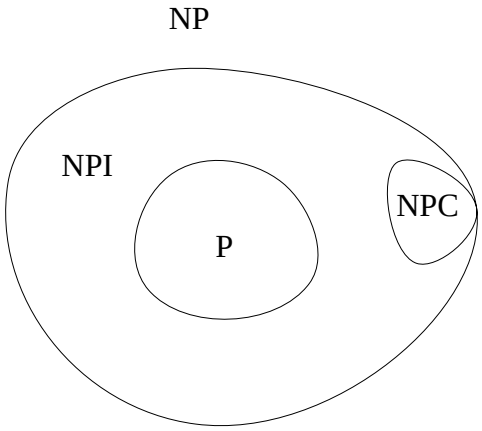


Figure 3.3: The map of **NP**, assuming that **NP**$\neq$ **P**. **NPI** can be further subdivided into an infinite hierarchy of problem classes by the technique used in showing that **NPI** exists.

$\mathbf{NP}^{\pi}$ is the class of all problem that can be solved in polynomial time by a non-deterministic Turing machine with access to an oracle deciding π. This notation is unfortunately somewhat misleading. It is possible to prove that there exists an oracle π_1 such that $\mathbf{NP}^{\pi_1} = \mathbf{P}^{\pi_1}$, but there is also an oracle π_2 such that $\mathbf{NP}^{\pi_2} \neq \mathbf{P}^{\pi_2}$ If π is satisfiability or another **NP**-complete problem, we arrive at the class $\mathbf{P}^{\mathbf{NP}}$ or $\Delta_2\mathbf{P}$. This is the first class in the polynomial hierarchy of classes that are harder than **NPC**. Spin game models that are in various classes of the polynomial hierarchy have been studied in [71].

A very nice survey of important computational complexity results from the last decade can be found in Fortnow's amusing paper [24].

3.8 Other models of computation — universality

In addition to the Turing machine, several other models of computation have been proposed. The Church-Turing hypothesis states that it does not matter which of these we use — they are all equivalent. If a function can be computed with one of them, it can also be computed with the others. Furthermore, the thesis also claims that all reasonable models of computation give rise to the same set of computable functions. The Church-Turing hypothesis can not be proven, but there is compelling evidence for its validity in the fact that all reasonable models of computation can in fact simulate each other, albeit with large slowdowns. Some other models of computation are the recursive functions, Post's string rewriting systems, and the λ-calculus. In the Post system [63], rules are given for rewriting strings based only on their first symbol — it is quite amazing that this simple system is capable of universal computation. The λ-calculus [17] is the basis for modern functional programming languages — it consists of function definitions and applications.

There has recently been much interest in so called quantum computers. For some problems [68], it has been shown that a quantum computer is more powerful than a classical, but none of these problems are known to be **NP**-complete. It is amusing to note that non-physical quantum computers (that utilise e.g., non-linear quantum mechanics) can solve **NPC** problems in polynomial time (e.g., [1, 29]).

An interesting extension of the idea of the Turing machines has been taken by Pudlák [64],

who considers populations of Turing machines that undergo genetic evolution. This is a model of parallel computation, and is more powerful than a single normal Turing machine. The set of all problems computable in polynomial time on such machines is equal to the set of all problems computable in polynomial space on a classical Turing machine, **PSPACE**. This class is strongly believed to be stricly larger than **P** for classical Turing machines, although this has not yet been proven.

DNA computers have also been used to solve some **NP** problems in what seems to be polynomial time [5]. Here the trick to solving seemingly exponential problems is massive parallelism. We fill a container with DNA strings representing all possible solutions to some problem. Using chemistry the DNA strings representing correct solutions are then separated from the others. A problem here is the difficulty of retrieving the correct solution if there are not many of them. This is exactly the same difficulty that prevents the implementation of a quantum λ-calculus [49].

4 The physics of constraint satisfaction problems

A general constraint satisfaction problem, as defined above, can be seen as a spin glass problem. There is growing interest among physicists in studying the physics of constraint satisfaction problem. In addition to the help that computer science can get from this, it is also possible that the artificial disordered systems studied here can provide new insights into the study of physical disordered systems.

An obvious method to solve an **NPC** problem is to search through all possible variable assignments until one is found that solves the problem. In some cases, such as **Horn-SAT** or 2-**SAT**, the problem structure is such that there are more efficient methods to construct a solution, but these are exceptions. The search methods work by assigning one variable at a time and then solving the resulting subproblem recursively. When it reaches a problem that has no solution, it must backtrack and try other assignments. The space that must be searched can be seen as a tree, where the internal nodes correspond to partial assignment and the leafs to complete assignments of all the variables in the problem.

In order to construct an efficient search method, it is important to prune the search tree, i.e., discover quickly if a subtree does not possess any leafs that are solutions to the problem. In the worst case, the solution is in the last leaf that is examined; this is the reason why the worst-case complexity is exponential.

The most interesting feature of these search methods for physicists is that they have been shown to give rise to threshold phenomena very similar to those occuring at physical phase transitions [39, 16, 33]. For some optimisation problem, it is possible to define a constrainedness parameter so that a randomly choosen problem instance that has a low degree of constrainedness is always solvable, while one that is highly constrained never has a solution. This is of course in a sense trivial, but it is surprising that the boundary between the two cases is sharp. Kirkpatrick and Selman [39] have shown that this transition sharpens as problem size is increased and that finite size scaling can be used to describe it. Friedgut [25, 2] has also shown rigourously that there is a sharp transition for all problem sizes, but there is not yet any proof that the transition happens for the same parameter value for different sizes. Note that some of the methods commonly used to generate more complicated random constraint satisfaction problems have been shown not to have a transition in the thermodynamic limit [4].

In addition, there is also a transition in the difficulty of finding a solution or showing that none exist. Is it very easy to find a solution for underconstrained problems — since most variable assignments do not lead to conflicts with others, not much backtracking will be needed. For overconstrained problems, on the other hand, the increased number of constraints makes the search methods quickly run into inconsistencies and not many nodes of the search tree will have to be examined. For problems in the region between over and underconstrained (termed critically constrained), the search method will have to spend a long time searching through dead ends that it can avoid in the other phases.

By treating all the constraints in the problem as independent, it is possible to make an ap-

proximation for the number of solutions of a problem with $M = \alpha N$ contraints and N variables. For simplicity, consider k-**SAT**. Each constraint here involves k variables and forbids one of the 2^k possible combinations of assignments to these variables. Approximate the probability that a constraint is violated in an assignment by $p_v = \frac{1}{2^k}$. Assuming that the constraints are independent then gives $(1 - p_v)^M$ as the probability of a formula with M clauses having no clause that is violated. Multiplying with the number of possible assignments, 2^N, then gives an approximation to the number of solutions for k-**SAT**

$$N_{\mathrm{sol}} = 2^N (1 - \frac{1}{2^k})^{\alpha N}. \tag{4.1}$$

To approximately determine the location of the phase transition, solve for α in $N_{\mathrm{sol}} = 1$, giving

$$\alpha_c = -\frac{\ln 2}{\ln (1 - \frac{1}{2^k})}. \tag{4.2}$$

For $k = 3$, this gives $\alpha_c = 5.17$, which is below the true value of $\alpha_c \approx 4.17$. For more information on various attempts to determine the location of the phase transition analytically, the reader is referred to paper I and references therein.

This gives qualitative explanations for both the solvable-unsolvable transition and the easy-hard-easy pattern of the amount of resources necessary to solve the problem. Mammen and Hogg [45] have recently found that the size of the smallest minimal unsolvable subproblems shows a behaviour that coincides roughly with that of the search cost, and also that search cost appears to be a strictly increasing function of this size. A minimum unsolvable subproblem is a subset of the problem that is unsolvable but becomes solvable if any variable and the constraints in which it appears are deleted from the problem.

Walsh [74] has made an interesting comparison between search methods for constraint satisfaction problems and renormalisation group flows from the theory of critical phenomena. Using a variant of the Davis-Putnam algorithm for satisfiability, Walsh has studied how the constrainedness changes during the search. The Davis-Putnam algorithm is a complete search algorithm. It works by visiting the nodes in the search tree described above. Since the formula used thus changes during the search procedure, the constrainedness can be calculated for each visited node in the search tree.

By plotting the constrainedness as a function of search depth and for different initial values of α, an interesting picture is found. For problems that are critically constrained, the constrainedness does not vary much as search progresses. For overconstrained problems, the constrainedness increases rapidly, while for underconstrained problems it decreases just as rapidly. That is, the constrainedness parameter α shows much the same behaviour as the coupling constant of a critical system. Here, starting at the critical coupling temperature means that the coupling constant is constant, while starting above or below the critical temperature will cause the coupling constant to be drawn towards either the high or low temperature fixed point representing the disordered and ordered phase, respectively. The comparison is of course to be expected, but is nonetheless interesting, since it provides a qualitative comparison between search procedures and renormalisation group flows.

The k-**SAT** problem has been studied in detail by Monasson and Zechhina [57, 56, 58, 60, 59] who have found interesting analogies between it and random field models. Among other things, they have found that the entropy stays finite at the transition. Studying a model that interpolates between 2-**SAT** and 3-**SAT** (by introducing a fraction p of clauses with three literals into a 2-**SAT** formula), they have also found that the transition changes character for a finite

$p \approx 0.413$ (see also [3]). This is discouraging news for those who think that statistical physics can provide a definite answer to the **P**=**NP** question, since the problem becomes **NP**-complete for any infinitesimal p. A good introduction to both the satisfiability problem, some algorithms used to solve it, and the phase transition phenomenon found in it can be found in the introductory article by Hayes [30]. The survey paper by Cook and Mitchell [18] contains a nice overview of the satisfiability problem, the algorithms used to solve it, and the threshold phenomena found for k-**SAT** from the point of view of a computer scientist.

One of the problems that has been studied most by physicists is the travelling salesperson problem. It too has been shown to exhibit a transition in solvability [27]. For a nice introduction to a physicist's view of this problem as well as a list of references, see [62].

5 The papers

In paper I we study the relaxation of the energy, defined as the number of unsatisfied constraints, of k-**COL** and k-**SAT**. We find that there is a change from fast, exponential decay to power law relaxation as the constrainedness (measured by the connectivity γ in k-**COL** and by the ratio of the number of clauses to the number of variables α in k-**SAT**) is increased. We also find evidence for a freezing transition: above a certain γ_c the $T = 0$ Monte Carlo method can no longer find a state with vanishing energy. Evidence is presented that the former transition exhibits finite-size scaling behaviour. Both the changes occur for smaller values of γ and α than the solvability transition. In addition, some data for the fraction of persistent spins $r(t)$ in k-**SAT** are presented. These data seem to indicate that there is a transition also in the behavior of $r(t)$.

Paper II deals with a simpler version of k-**COL**. Here Ising spin models on random graphs with connectivity γ but with only ferromagnetic interactions are studied. For this model the $T = 0$ ground state is trivial for all γ, but a freezing transition for the MC algorithm is nevertheless found. There exists a range of γ for which the model freezes, but for small and large average degrees there is an exponentially fast decay. There is a smaller range of connectivities for which this freezing persists even using simulated annealing. The related model with conserved magnetism (which is a model for graph partitioning, another **NPC**-problem) displays exponential relaxation and freezing, and here there is no improvement using simulated annealing.

"What was it, in the end?"
"What it always is. A handful of yarn; a little
weaving and stitching; some embroidery perhaps.
A few loose ends, but that's only to be expected."
— Clotho and Lachesis, in The Kindly Ones

References

[1] D. S. Abrams and S. Lloyd. Nonlinear quantum mechanics implies polynomial-time solution for NP-complete and #P problems. Report No. quant-ph/9801041.

[2] D. Achlioptas and E. Friedgut. A Sharp Threshold for k-Colorability. Manuscript, 1997. Available at **http://www.ma.huji.ac.il/∼ehudf/**.

[3] D. Achlioptas, L. M. Kirousis, E. Kranakis, and D. Krizanc. Rigorous Results for Random $(2 + p)$-SAT. Manuscript 1997.

[4] D. Achlioptas, L. M. Kirousis, E. Kranakis, D. Krizanc, M. S. O. Molloy, and Y. C. Stamatiou. Random Constraint Satisfaction: A More Accurate Picture. In *Proceedings of Third International Conference on Principles and Practice of Constraint Programming*, volume 1330, pages 107–120, Berlin, 1997. Springer.

[5] L. M. Adleman. Molecular Computation of Solutions to Combinatorial Problems. *Science*, 266(1021-1024), 1994.

[6] P. W. Anderson. Reference Frame articles in *Physics Today* January, March, June, and September 1988, July and September 1989, and March 1990.

[7] P. W. Anderson. Spin Glass Hamiltonians: A Bridge Between Biology, Statistical Mechanics and Computer Science. In D. Pines, editor, *Emerging Syntheses in Science*, pages 17–20. Addison-Wesley, Reading, Ma, 1984.

[8] K. Appel and W. Haken. Every planar map is four colorable. Part I. Discharging. *Illinois J. Math.*, 21:429–490, 1977.

[9] K. Appel, W. Haken, and J. Koch. Every planar map is four colorable. Part II. Reducibility. *Illinois J. Math.*, 21:491–567, 1977.

[10] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. on Computing*, 26(5):1411–1473, 1997.

[11] K. Binder and A. P. Young. Spin glasses: Experimental facts, theoretical concepts, and open questions. *Rev. Mod. Phys.*, 58(4):801–976, 1986.

[12] J. Binney, N. Dowrick, A. Fisher, and M. Newman. *The Theory of Critical Phenomena*. Oxford University Press, 1992.

[13] S. Boettcher and A. Percus. Nature's Way of Optimizing. Report No. cond-mat/9901351.

[14] B. Bollobás. *Extremal Graph Theory*. Academic Press, New York, 1978.

[15] P. Chaikin and T. C. Lubensky. *Principles of condensed matter physics*. Cambridge University Press, 1995.

[16] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the Really Hard Problems Are. In *Proceedings of IJCAI-91*, pages 331–337, San Mateo, CA, 1991. Morgan Kauffman.

[17] A. Church. *The Calculi of Lambda-Conversion*, volume 6 of *Annals of Mathematics Studies*. Princeton University Press, 1941.

[18] S. A. Cook and D. G. Mitchell. Finding Hard Instances of the Satisfiablility Problem. Manuscript 1998.

[19] G. A. Cowan, D. Pines, and D. Meltzer, editors. *Complexity: Metaphors, Models, and Reality*, volume XIX of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, Reading, Ma, 1994.

[20] J. P. Crutchfield and D. P. Feldman. Statistical Complexity of Simple 1D Spin Systems. *Phys. Rev. E*, 55(2):1239R–1243R, 1997.

[21] A. Dewdney. *The Turing Omnibus*. Computer Science Press, Rockville, 1989.

[22] D. P. Feldman and J. P. Crutchfield. Measures of Statistical Complexity: Why? *Phys. Lett. A*, 238:244–252, 1998.

[23] R. P. Feynman. *Feynman Lectures on Computation*. Addison-Wesley, Reading, Ma, 1996.

[24] L. Fortnow. My Favorite Ten Complexity Theorems of the Past Decade. ECCC Report 94-021.

[25] E. Friedgut. Necessary and Sufficient Conditions for Sharp Thresholds of Graph Properties, and the k-SAT Problem. Manuscript, 1997. Available at **http://www.ma.huji.ac.il/~ehudf/**.

[26] M. R. Garey and D. S. Johnson. *Computers and Intractability. A guide to the Theory of NP-Completeness*. W H Freeman, New York, 1979.

[27] I. P. Gent and T. Walsh. The TSP phase transition. *Artificial Intelligence*, 88:349–358, 1996.

[28] R. J. Glauber. Time-Dependent Statistics of the Ising Model. *J. Math. Phys*, 4:294, 1963.

[29] P. Gossett. NP in BQP with Nonlinearity. Report No. quant-ph/9804025.

[30] B. Hayes. Can't Get No Satisfaction. *American Scientist*, 85(2):108–112, 1998.

[31] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation*. Addison-Wesley, 1991.

[32] T. Hogg. Applications of Statistical Mechanics to Combinatorial Search Problems. In *Annual Reviews of Computational Physics*, volume 2, pages 357–406, Singapore, 1995. World Scientific.

[33] T. Hogg, B. A. Huberman, and C. P. Williams. Editorial: Phase transitions and the search problem. *Artificial Intelligence*, 81:1–15, 1996.

[34] P. C. Hohenberg and B. I. Halperin. Theory of dynamic critical phenomena. *Rev. Mod. Phys.*, 49(3):435, 1977.

[35] J. Houdayer and O. C. Martin. Renormalisation for Discrete Optimization. Report No. cond-mat/9901276.

[36] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation; part II, Graph Coloring and Number Partitioning. *Operations Research*, 39(3):378–406, 1991.

[37] K. Kawasaki. In C. Domb and M. S. Green, editors, *Phase Transitions and Critical Phenomena*, volume 2. Academic Press, London, 1972.

[38] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[39] S. Kirkpatrick and B. Selman. Critical Behavior in the Satisfiability of Random Boolean Expression. *Science*, 264:1297–1301, May 1994.

[40] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, Ma, 2 edition, 1998.

[41] Z. W. Lai, G. F. Mazenko, and O. T. Valls. Classes for growth kinetics problems at low temperatures. *Phys. Rev. B*, 37(16):9481–9494, 1988.

[42] J. S. Langer. An introduction to the kinetics of first-order phase transition. In C. Godrèche, editor, *Solids far from equilibrium*, pages 297–364. Cambridge University Press, 1992.

[43] C. G. Langton, editor. *Artificial Life*. Santa Fe Institute Studes in the Sciences of Complexity. Addison-Wesley, Reading, Ma, 1988.

[44] D. B. Lenat. The nature of heuristics. *Artificial Intelligence*, 19:189–249, 1982.

[45] D. L. Mammen and T. Hogg. A New Look at the Easy-Hard-Easy Pattern of Combinatorial Search Difficulty. *J. of Artificial Intelligence Research*, 7:47–66, 1997.

[46] J. C. Martin. *Introduction to languages and the theory of computation*. McGraw-Hill, New York, 2nd edition, 1997.

[47] Y. Matiyasevich. Primes are nonnegative values of a polynomial in ten variables. *Zapiski Sem. Leningrad Mat. Inst. Steklov*, 68:62–82, 1977. Translated in Journal of Soviet Mathematics **15** 33-44 (1981).

[48] Y. Matiyasevich. *Hilbert's Tenth Problem*. The MIT Press, Cambridge, 1993. Excerpts available at
www.informatik.uni-stuttgart.de/ifi/ti/personen/Matiyasevich/H10Pbook/index.html.

[49] P. Maymin. Programming Complex Systems. Report No. quant-ph/9710035.

[50] W. McCune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263–276, 1997. See also **http://www-unix.mcs.anl.gov/ mccune/papers/robbins/**.

[51] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. *J. Chem. Phys*, 21:1087, 1953.

[52] M. Mezard. On the ubiquity of spin glass concepts and methods. *Physica A*, 200:111–117, 1993.

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

[54] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, N.J., 1967.

[55] D. G. Mitchell. Hard Problems for CSP Algorithms. To appear in Proceedings, AAAI-98.

[56] R. Monasson. Optimisation problems and replica symmetry breaking in finite connectivity spin-glasses. *J. Phys. A*, 31(2):513–29, 1998.

[57] R. Monasson. Some remarks on hierarchical replica symmetry breaking in finite-connectivity systems. *Phil. Mag. B*, 77(5):1515–1521, 1998.

[58] R. Monasson and R. Zecchina. Tricritical Points in Random Combinatorics: the (2+p)-SAT case. Report No. cond-mat/9810008.

[59] R. Monasson and R. Zecchina. Entropy of the K-Satisfiability Problem. *Phys. Rev. Lett.*, 76(21):3881–3885, 1996.

[60] R. Monasson and R. Zecchina. Statistical Mechanics of the Random K-satisfiability Model. *Phys. Rev. E*, 56(2):1357–1370, 1997.

[61] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

[62] A. Percus. *The Traveling Salesman and Related Stochastic Problems*. PhD thesis, Université Paris 6, 1997.

[63] E. Post. Finite combinatory processes-formulation, I. *J. of Symbolical Logic*, 1:103–105, 1936.

[64] P. Pudlák. Complexity Theory and Genetics. ECCC Report 94-013.

[65] M. Rickert and K. Nagel. Experiences with a Simplified Microsimulation for the Dallas/Fort Worth Area. SFI Working Paper 97-03-027.

[66] Y. Rogozhin. Small universal Turing machines. *Theoretical Computer Science*, 168:215–240, 1996.

[67] A. Sadiq and K. Binder. Dynamics of Formation of Two-Dimensional Ordered Structures. *J. Stat. Phys*, 35(5-6):517–585, 1984.

[68] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proc. 35th Symp. on Foundations of Computer Science*, pages 124–134, 1994.

[69] J. D. Shore, M. Holzer, and J. P. Sethna. Logarithmically slow domain growth in non-randomly frustrated systems: Ising models with competing interactions. *Phys. Rev. B*, 46(18):11376–11404, 1992.

[70] P. M. A. Sloot, A. Schoneveld, J. F. de Ronde, and J. A. Kaandorp. Large-Scale Simulations of Complex System, Part I: Conceptual Framework. Santa Fe Institute Working Paper 97-07-070.

[71] P. Varga. Minimax games, spin glasses and the polynomial-time hierarchy of complexity classes. *Phys. Rev. E*, 57(6):6487–92, 1998.

[72] T. Vojta. In an Ising model with spin-exchange dynamics damage always spreads. Report No. cond-mat/9803053.

[73] T. Vojta and M. Schreiber. An unexpected difference between regular and random order of updates in damage spreading simulations. Report No. cond-mat/9807229.

[74] T. Walsh. The Constrainedness Knife-Edge. Available at **http://www.cs.strath.ac.uk/∼apes/apepapers.html**.

[75] H. Wang. *Popular Lectures on Mathematical Logic*. Dover, New York, 1981.

[76] P. H. Winston. *Artificial Intelligence*. Addison-Wesley, Reading, Ma, 2nd edition, 191984.

[77] D. H. Wolpert, editor. *The Mathematics of Generalization*, volume XX of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, Reading, Ma, 1994.

[78] A. P. Young, editor. *Spin Glasses and Random Fields*. World Scientific, Singapore, 1998.

Paper I:

Paper II: