COMPUTATIONAL COMPLEXITY FOR PHYSICISTS

The theory of computational complexity has some interesting links to physics, in particular to quantum computing and statistical mechanics. This article contains an informal introduction to this theory and its links to physics.

Compared to the traditionally close relationship between physics and mathematics, an exchange of ideas and methods between physics and computer science barely exists. However, the few interactions that have gone beyond Fortran programming and the quest for faster computers have been successful and have provided surprising insights in both fields. This is particularly true for the mutual exchange between statistical mechanics and the theory of *computational complexity*. Here, I discuss this exchange in a manner directed at physicists with little or no knowledge of the theory.

The measure of complexity

The branch of theoretical computer science known as computational complexity is concerned with classifying problems according to the computational resources required to solve them (for additional information about this field, see the "Related Works" sidebar). What can be measured (or computed) is the time that a particular algorithm uses to solve the problem. This time, in turn, depends on the algorithm's implementation as well as the computer on which the program is running.

The theory of computational complexity provides us with a notion of complexity that is largely independent of implementation details and the computer at hand. Its precise definition requires a considerable formalism, however. This is not surprising because it is related to a highly nontrivial question that touches the foundation of mathematics: *What do we mean when we say a problem is solvable?* Thinking about this question leads to Gödel's incompleteness theorem, Turing machines, and the Church-Turing thesis on computable functions.

Here we adopt a more informal, pragmatic viewpoint. A problem is solvable if a computer program written in your favorite programming language can solve it. Your program's running time or *time complexity* must then be defined with some care to serve as a meaningful measure of the problem's complexity.

Time complexity

In general, running time depends on a problem's size and on the specific input data—the *instance*. Sorting 1,000 numbers takes longer than sorting 10 numbers. Some sorting algorithms run faster if the input data is partially sorted already. To minimize the dependency on the specific instance, we consider the *worst-case* time complexity $T(n)$:

1521-9615/02/$17.00 © 2002 IEEE

STEPHAN MERTENS
Otto-von-Guericke University, Magdeburg

Related Work

For introductions into the field of computational complexity, see

E.L. Lawler et al., eds., *The Traveling Salesman Problem*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, New York, 1985.

H.R. Lewis and C.H. Papadimitriou, "The Efficiency of Algorithms," *Scientific American*, vol. 109, no. 1, Jan. 1978, pp. 96–109.

C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice-Hall, Englewood Cliffs, N.J., 1982.

For a deep understanding of the field, read the following classic textbooks:

M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, New York, 1997.

C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, Mass., 1994.

The complexity of Ising spin systems is discussed in:

B. Hayes, "Computing Science: The World in a Spin," *American Scientist*, vol. 88, no. 5, Sept./Oct. 2000, pp. 2384–388; www.amsci.org/amsci/issues/comsci00/compsci2000-09.html.

For an entertaining introduction (written by a physicist) to Gödel's incompleteness theorem, Turing machines, and the Church-Turing Thesis on computable functions, read

R. Penrose, *The Emperor's New Mind*, Oxford Univ. Press, 1989.

To learn more about quantum parallelism, see www.qubit.org or

D. Aharonov, "Quantum Computation," *Ann. Rev. Computational Physics VI*, D. Stauffer, ed., World Scientific, 1998.

Phase transitions in computational complexity are discussed in:

O. Dubois et al., eds., "Phase Transitions in Combinatorial Problems," special issue of *Theor. Comp. Sci.*, vol. 265, nos. 1–2, 2001.

B. Hayes, "Computing Science: The Easiest Hard Problem,"*American Scientist*, vol. 90, no. 2, Mar./Apr. 2002, pp. 113–117; www.amsci.org/amsci/issues/comsci02/compsci2002-03.html.

B. Hayes, "Computing Science: Can't Get No Satisfaction," *American Scientist*, vol. 85, no. 2, Mar./Apr. 1997, pp. 108–112; www.amsci.org/amsci/issues/Comsci97/compsci9703.html.

$$T(n) = \max_{|x|=n} t(x), \qquad (1)$$

where $t(x)$ is the algorithm's running time for input data x (in arbitrary units), and the maximum is taken over all problem instances of size n. The worst-case time is an upper bound for the observable running time.

A measure of time complexity should be based on a unit of time that is independent of the specific CPU's clock rate. Such a unit is provided by the time it takes to perform an elementary operation such as adding two integer numbers. Measuring the time in this unit means counting the number of elementary operations your algorithm executes. This number in turn depends strongly on the algorithm's implementation details—smart programmers and optimizing compilers will try to reduce it. Therefore, rather than consider the precise number $T(n)$ of elementary operations, we only consider the asymptotic behavior of $T(n)$ for large values of n as the Landau symbols $\mathcal{O}$ and Θ denote:

- We say $T(n)$ is of order at most $g(n)$ and write $T(n) = \mathcal{O}(g(n))$ if positive constants c and n_0 exist such that $T(n) \le cg(n)$ for all $n \ge n_0$.
- We say $T(n)$ is of order $g(n)$ and write $T(n) = \Theta(g(n))$ if positive constants c_1, c_2, and n_0 exist such that $c_1 g(n) \le T(n) \le c_2 g(n)$ for all $n \ge n_0$.

Multiplying two $n \times n$ matrixes requires n^3 multiplications, according to the textbook formula. However, this does not mean that the problem of multiplying two $n \times n$ matrices has complexity $\Theta(n^3)$. The textbook formula is a particular algorithm, and an algorithm's time complexity is only an upper bound for a problem's inherent complexity. In fact, researchers have found faster matrix multiplication algorithms with complexity $\mathcal{O}(n^\alpha)$ and $\alpha < 3$ during the last decades—the current record being $\alpha = 2.376$.[1] Because the product matrix has n^2 entries, α cannot be smaller than 2; it is an open question whether an algorithm can achieve this lower bound.

A problem where the upper bound from algorithmic complexity meets an inherent lower bound is sorting n items. Under the general assumption that comparisons between pairs of items are the only source of information about them, $\Theta(n \log n)$ is a lower bound for the number of comparisons to sort n items in the worst case.[2] This bound is met by algorithms such as "heapsort" or "mergesort."

Problem size

Our measure of time complexity still depends on the somewhat ambiguous notion of problem size. In the matrix multiplication example, we tacitly took the number n of rows of one input matrix as the "natural" measure of size. Using the number of elements $m = n^2$ instead speeds up the $\mathcal{O}(n^3)$ algorithm to $\mathcal{O}(m^{3/2})$ without changing a single line of program code. An unambiguous definition of problem size is required to compare algorithms.

In computational complexity, all problems solvable by a polynomial algorithm—that is, an algorithm with time complexity $\Theta(n^k)$ for some k—are lumped together and called *tractable*. Problems that can only solvable by algorithms with nonpolynomial running time, such as $\Theta(2^n)$ or $\Theta(n!)$, are also lumped together and called *intractable*. There are practical as well as theoretical reasons for this rather coarse classification. One of the theoretical advantages is that it does not distinguish between the $\mathcal{O}(n^3)$ and $\mathcal{O}(m^{3/2})$ algorithm example from above; hence, we can afford some sloppiness and stick with our ambiguous natural measure of problem size.

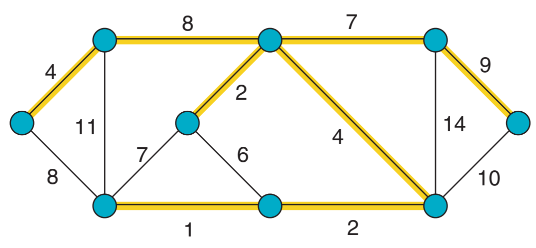

Figure 1. A weighted graph and its minimum spanning tree (shaded edges).

Tractable and intractable problems

The terms tractable and intractable for problems with polynomial and exponential algorithms refer to the fact that an exponential algorithm means a hard limit for the accessible problem sizes. Suppose that, with your current equipment, you can solve a problem of size n just within your schedule. If your algorithm has complexity $\Theta(2^n)$, a problem of size $n + 1$ needs twice the time, pushing you off schedule. The increase in time caused by an $\Theta(n)$ or $\Theta(n^2)$ algorithm, on the other hand, is far less dramatic and can easily be compensated for by upgrading your hardware.

You might argue that a $\Theta(n^{100})$ algorithm outperforms a $\Theta(2^n)$ algorithm only for problem sizes that will never occur in your application. A polynomial algorithm for a problem usually goes hand in hand with a *mathematical insight* into that problem, which lets you find a polynomial algorithm with a small degree, typically $\Theta(n^k)$, $k = 1$, 2, or 3. Polynomial algorithms with $k > 10$ are rare and arise in rather esoteric problems. It is this mathematical insight (or rather the lack of it) that turns intractable problems into a challenge for algorithm designers and computer scientists. Let's look at a few examples of tractable and intractable problems to learn more about what separates one from the other.

Tractable trees

Consider the following problem from network design. You have a business with several offices, and you want to lease phone lines to connect them. The phone company charges different amounts to connect different pairs of cities, and your task is to select a set of lines that connects all your offices with a minimum total cost.

In mathematical terms, the cities and the lines between them form the vertices V and edges E of a weighted graph $G = (V, E)$, the weight of an edge being the corresponding phone line's leasing costs. Your task is to find a subgraph that connects all vertices in the graph (for example, a spanning subgraph) and whose edges have minimum total weight. Your subgraph should not contain cycles because you can always remove an edge from a cycle, keeping all nodes connected and reducing the cost. A graph without cycles is a tree, so what you are looking for is a minimum spanning tree (MST) in a weighted graph (see Figure 1). So, given a weighted graph $G = (V, E)$ with nonnegative weights, find a spanning tree $T \subseteq G$ with minimum total weight.

How do you find an MST? A naive approach is to generate all possible trees with n vertices and keep the one with minimal weight. The enumeration of all trees can be done using Prüfer codes,[3] but Cayley's formula tells us that there are n^{n-2} different trees with n vertices. Already for $n = 100$ there are more trees than atoms in the observable universe. Hence, exhaustive enumeration is prohibitive for all but the smallest trees. The mathematical insight that turns MST into a tractable problem is this:

Lemma: Let $U \subset V$ be any subset of the vertices of $G = (V, E)$, and let e be the edge with the smallest weight of all edges connecting U and $V - U$. Then e is part of the MST.

Proof (by contradiction): Suppose T is an MST not containing e. Let $e = (u, v)$, with $u \in U$ and $v \in V - U$. Then, because T is a spanning tree, it contains a unique path from u to v that together with e forms a cycle in G. This path must include another edge f connecting U and $V - U$. Now $T + e - f$ is another spanning tree with less total weight than T. So T was not an MST.

The lemma lets an MST grow edge by edge—using Prim's algorithm, for example:

```
Prim($G$)
**Input**: weighted graph $G(V, E)$
**Output**: minimum spanning tree $T \subseteq G$
```

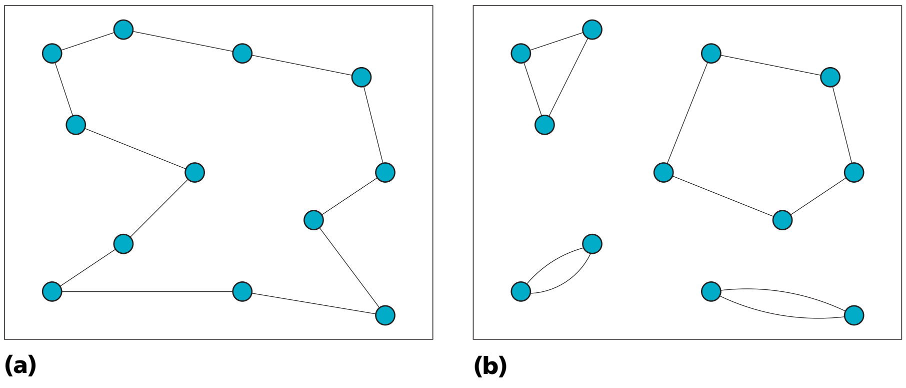

Figure 2. Same instance, different problems: A valid configuration of (a) the Traveling Salesman Problem and (b) the Assignment Problem. Whereas the Assignment Problem can be solved in polynomial time, the TSP is intractable.

```
begin
    Let $T$ be a single vertex $v$ from $G$
    while $T$ has less than $n$ vertices
        find the minimum edge
          connecting $T$ to $G - T$
        add it to $T$
    end
end
```

The precise time complexity of Prim's algorithm depends on the data structure used to organize the edges, but in any case, $\mathcal{O}(n^2 \log n)$ is an upper bound (faster algorithms are discussed elsewhere[3]). Equipped with such a polynomial algorithm, you can find MSTs with thousands of nodes within seconds on a personal computer. Compare this to an exhaustive search. According to our definition, finding an MST is a tractable problem.

Intractable itineraries

Encouraged by the efficient algorithm for an MST, let us now investigate a similar problem. Your task is to plan an itinerary for a traveling salesman who must visit n cities. You are given a map with all cities and the distances between them. In what order should the salesman visit the cities to minimize the total distance traveled? You number the cities arbitrarily and write down the distance matrix (d_{ij}), where d_{ij} denotes the distance between city i and city j. A tour is given by a *cyclic permutation* π: $[1...n] \to [1...n]$, where $\pi(i)$ denotes the successor of city i. Your problem can be defined as

The *Traveling Salesman Problem* (TSP): Given an $n \times n$ distance matrix with elements $d_{ij} \geq 0$, find a cyclic permutation $\pi : [1...n] \to [1...n]$ that minimizes

$$c_n(\pi) = \sum_{i=1}^{n} d_{i\pi(i)}. \tag{2}$$

The TSP is probably the most famous optimization problem (see www.keck.caam.rice.edu/tsp). Finding good solutions, even to large problems, is not difficult, but how can we find the best solution for a given instance? There are $(n - 1)!$ cyclic permutations, and calculating the length of a single tour takes $\mathcal{O}(n)$. So, an exhaustive search has complexity $\mathcal{O}(n!)$. Again this approach is limited to very small instances.

Is there a mathematical insight that provides a shortcut to the optimum solution, such as for an MST? Nobody knows. Despite many efforts, researchers have not found a polynomial algorithm for the TSP. There are some smart and efficient (that is, polynomial) algorithms that find good solutions but do not guarantee yielding the optimum solution.[4] According to this definition, the TSP is intractable.

Why is the TSP intractable? Again, nobody knows. There is no proof that excludes the existence of a polynomial algorithm for the TSP, so maybe someday somebody will come up with a polynomial algorithm and the corresponding mathematical insight. This is very unlikely, however, as we will see soon.

The TSP's intractability astonishes all the more considering the tractability of a very similar, almost identical problem, the Assignment problem:

Assignment: Given an $n \times n$ cost matrix with elements $d_{ij} \geq 0$, find a permutation $\pi : [1...n] \to [1...n]$ that minimizes

$$c_n(\pi) = \sum_{i=1}^{n} d_{i\pi(i)}. \tag{3}$$

The only difference between the TSP and Assignment is that the latter allows all permutations on n items, not just the cyclic ones. If d_{ij} denotes distances between cities, Assignment corresponds to total tour length minimization for a variable number of salesmen, each traveling his own *subtour* (see Figure 2).

The classical application of Assignment is the assignment of n tasks to n workers, subject to the constraint that each worker is assigned exactly

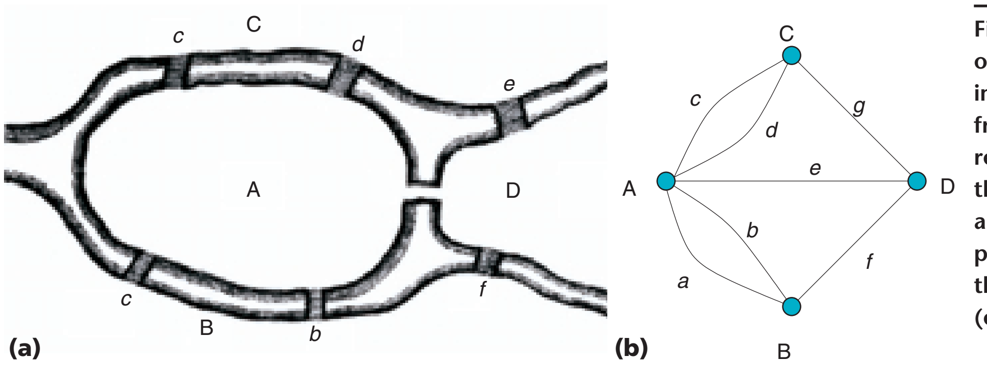

Figure 3. The seven bridges of Königsberg, (a) as drawn in Leonhard Euler's paper from 1736[6] and (b) represented as a graph. In the graph, the riverbanks and islands are condensed to points (vertices), and each of the bridges is drawn as a line (edge).

one task. Let d_{ij} denote the cost of worker I performing task j, and $\pi(i)$ denote the task assigned to worker i. Assignment is the problem of minimizing the total cost.

There are $n!$ possible assignments of n tasks to n workers—hence, exhaustive enumeration is again prohibitive. In contrast to the TSP, however, we can solve Assignment in polynomial time—for example, using the $\mathcal{O}(n^3)$ Hungarian algorithm.[5] Compared to an MST, the algorithm and the underlying mathematical insight are a bit more involved and not discussed here.

Decision problems

So far, we have discussed optimization problems: solving MST, TSP, or Assignment implies that we compare an exponential number of feasible solutions with each other and pick the optimum solution. Exhaustive search does this explicitly; polynomial shortcuts implicitly. By investigating simpler problems whose solutions are recognizable without explicit or implicit comparison to all feasible solutions, we might learn more about the barrier that separates tractable from intractable problems. So, let's consider *decision problems*, whose solution is either "yes" or "no."

We can turn any optimization problem into a decision problem by adding a bound B to the instance. For example,

MST (decision): Given a weighted graph $G = (V, E)$ with nonnegative weights and a number $B \geq 0$, does G contain a spanning tree T with total weight $\leq B$?

TSP (decision): Given an $n \times n$ distance matrix with elements $d_{ij} \geq 0$ and a number $B \geq 0$, is there a tour π with length $\leq B$?

In a decision problem, the feasible solutions are not evaluated relative to each other but to an absolute criterion: a tour in the TSP either has length $\leq B$ or it doesn't.

We can solve MST(D) in polynomial time by simply solving the optimization variant MST and comparing the result to the parameter B. For the TSP(D), this approach does not help. In fact, we see later that a polynomial algorithm exists for the TSP(D) if and only if there exists a polynomial algorithm for the TSP. It seems as if we cannot learn more about the gap between tractable and intractable problems by considering decision variants of optimization problems. So let's look at other decision problems not derived from optimization problems.

Eulerian circuits

Our first genuine decision problem dates back to the 18th century, where in the city of Königsberg (now Kaliningrad), seven bridges crossed the river Pregel and its two arms (see Figure 3). A popular puzzle of the time asked if it were possible to walk through the city crossing each of the bridges exactly once and returning home.

Leonhard Euler solved this puzzle in 1736.[6] First, he recognized that to solve the problem, the only thing that matters is the pattern of interconnections of the banks and islands—a graph $G = (V, E)$ in modern terms. The graph corresponding to the puzzle of the Königsberg bridges has four vertices for the two banks and two islands and seven edges for the bridges (see Figure 3). Euler's paper on the Königsberg bridges is the birth of graph theory.

To generalize the Königsberg bridges problem, we need some terminology from graph theory.[3] A *walk* in a graph $G = (V, E)$ is an alternating sequence of vertices $v \in V$ and edges $(v, v') \in E$: $v_1, (v_1, v_2), v_2, (v_2, v_3), \ldots, (v_{l-1}, v_l), v_l$. Note that the sequence begins and ends with a vertex, and each edge is incident with the vertices immediately preceding and succeeding it. A walk is termed *closed* if $v_l = v_1$; otherwise, it is *open*. A walk is called a *trail* if all its edges are distinct, and a closed trail is called a *circuit*. What the strollers in Königsberg tried to find was a circuit

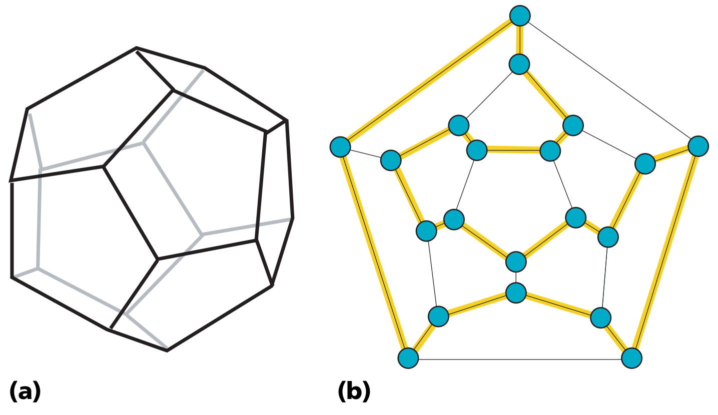

Figure 4. Sir Hamilton's Icosian game: (a) Find a route along the edges of the dodecahedron, passing each corner exactly once and returning to the starting corner. (b) A solution is indicated (shaded edges) in the planar graph that is isomorphic to the dodecahedron.

that contains all edges. In honor of Euler, such a circuit is called an *Eulerian circuit*.

We can now define the generalization of the Königsberg bridges problem:

Eulerian Circuit: Given a graph $G = (V, E)$, does G contain an Eulerian circuit?

Obviously, this is a decision problem. The answer is either "yes" or "no," and we can check to see whether a circuit is Eulerian without resorting to all possible circuits.

Once again, an exhaustive search would solve this problem, but Euler noticed the intractability of this approach. More than 200 years before the advent of computers, he wrote,

> The particular problem of the seven bridges of Königsberg could be solved by carefully tabulating all possible paths, thereby ascertaining by inspection which of them, if any, met the requirement. This method of solution, however, is too tedious and too difficult because of the large number of possible combinations, and in other problems where many more bridges are involved it could not be used at all.[7]

He solved the Königsberg bridges problem not by listing all possible trails but by using mathematical insight. He noticed that in a circuit, you must leave each vertex via an edge different from the edge that has taken you there. In other words, the vertex's degrees (that is, the number of edges adjacent to it) must be even. This is obviously a necessary condition, but Euler proved that it is also sufficient:

Theorem: A connected graph $G = (V, E)$ contains an Eulerian circuit if and only if the degree of every vertex $v \in V$ is even.

Euler's theorem lets us devise a polynomial algorithm for the Eulerian Circuit: Check the degree of every vertex in the graph. If one vertex has an odd degree, return "no." If all vertices have an even degree, return "yes." This algorithm's running time depends on the graph's encoding. If $G = (V, E)$ is encoded as a $|V| \times |V|$ adjacency matrix with entries a_{ij} = number of edges connecting v_i and v_j, the running time is $\mathcal{O}(|V|^2)$.

Thanks to Euler, the Eulerian Circuit is a tractable problem. The burghers of Königsberg, on the other hand, had to learn from Euler that they would never find a walk through their hometown crossing each of the seven bridges exactly once.

Hamiltonian cycles

Another decision problem is associated with the mathematician and Astronomer Royal of Ireland, Sir William Rowan Hamilton. In 1859, Hamilton put on the market a new puzzle called the Icosian game (see Figure 4).

Generalizing the Icosian game calls for some more definitions from graph theory: A closed walk in a graph is called a *cycle* if all its vertices (except the first and the last) are distinct. A Hamiltonian cycle is one that contains all vertices of a graph. The generalization of the Icosian game then reads

Hamiltonian Cycle: Given a graph $G = (V, E)$, does G contain a Hamiltonian cycle?

There is a certain similarity between the Eulerian Circuit and Hamiltonian Cycle. In the former, we must pass each edge once—in the latter, each vertex once. Despite this resemblance, the two problems represent entirely different degrees of difficulty. The available mathematical insights into the Hamiltonian Cycle provide us neither with a polynomial algorithm nor with a proof that such an algorithm is impossible. The Hamiltonian Cycle is intractable, and nobody knows why.

Coloring

Imagine we wish to arrange talks in a congress in such a way that no participant will be forced to miss a talk he or she wants to hear. Assuming a good supply of lecture rooms that lets us hold as many parallel talks as we like, can we finish the program within k time slots? We can formulate this problem in terms of graphs: Let G be a graph whose vertices are the talks and in which two talks are adjacent (joined by an edge) if and only if there is a participant wishing to attend both. Your task is to assign one of the k time slots to each vertex in such a way that adjacent vertices have different slots. The common formulation of this problem uses colors instead of time slots:

k-*Coloring*: Given a graph $G = (V, E)$, is there a coloring of the vertices of G using at most k different colors such that no two adjacent vertices have the same color?

When $k = 2$, this problem is tractable—constructing a polynomial algorithm is an easy exercise. For $k = 3$, however, things change considerably: 3-Coloring is intractable. Note that for larger k, the problem gets easier again: a *planar* graph is always colorable with four colors. This is the famous 4-Color Theorem. 3-Coloring remains intractable even when restricted to planar graphs.

Satisfiability

I close this section with a decision problem that is not from graph theory but from Boolean logic. A Boolean variable x can take on the value 0 (false) or 1 (true). Boolean variables can be combined in *clauses* using the Boolean operators

- NOT $\bar{\cdot}$ (negation): the clause $\bar{x}$ is true ($\bar{x} = 1$) if and only if x is false ($x = 0$).
- AND $\wedge$ (conjunction): the clause $x_1 \wedge x_2$ is true ($x_1 \wedge x_2 = 1$) if and only if both variables are true: $x_1 = 1$ and $x_2 = 1$.
- OR $\vee$ (disjunction): the clause $x_1 \vee x_2$ is true ($x_1 \vee x_2 = 1$) if and only if at least one of the variables is true: $x_1 = 1$ or $x_2 = 1$.

A variable x or its negation $\bar{x}$ is called a *literal*. Different clauses can combine to yield complex Boolean formulas such as

$$F_1(x_1,x_2,x_3) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \quad . \tag{4}$$

A Boolean formula evaluates to either 1 or 0, depending on the assignment of the Boolean variables. For example, $F_1 = 1$ for $x_1 = 1, x_2 = 1, x_3 = 0$, and $F_1 = 0$ for $x_1 = x_2 = x_3 = 1$. A formula F is called *satisfiable* if there is at least one assignment of the variables such that the formula is true. F_1 is satisfiable,

$$F_2(x_1,x_2) = (\bar{x}_1 \vee x_2) \wedge \bar{x}_2 \wedge x_1 \tag{5}$$

is not.

We can write every Boolean formula in *conjunctive normal form* (CNF)—that is, as a set of clauses C_k combined exclusively with the AND operator

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m, \tag{6}$$

where the literals in each clause are combined exclusively with the OR operator. The examples F_1 and F_2 are both written in CNF. Each clause can be considered a constraint on the variables, and satisfying a formula means satisfying a set of (possibly conflicting) constraints simultaneously. Hence, consider the following as a prototype of a constraint satisfaction problem:[8]

Satisfiability (SAT): Given disjunctive clauses $C_1, C_2, \ldots, C_m$ of literals, where a literal is a variable or negated variable from the set $\{x_1, x_2, \ldots, x_n\}$, is there an assignment of variables that satisfies all clauses simultaneously?

Fixing the number of literals in each clause leads to

k-*SAT*: Given disjunctive clauses $C_1, C_2, \ldots, C_m$ of k literals each, where a literal is a variable or negated variable from the set $\{x_1, x_2, \ldots, x_n\}$, is there an assignment of variables that satisfies all clauses simultaneously?

Polynomial algorithms are known for 1-SAT and 2-SAT.[9] No polynomial algorithm is known for general SAT and k-SAT if $k > 2$.

Complexity classes

Now we have seen enough examples to introduce two important complexity classes for decision problems and to discuss how these classes relate to other kinds of problems.

Tractable problems

Defining the class of *tractable decision problems* is easy: it consists of those problems for which a polynomial algorithm is known. The corresponding class is named $\mathcal{P}$ for "polynomial":

Definition: A decision problem P is element of the class $\mathcal{P}$ if and only if a polynomial time algorithm can solve it.

Eulerian Circuit, 2-Coloring, MST(D), and so forth are in $\mathcal{P}$.

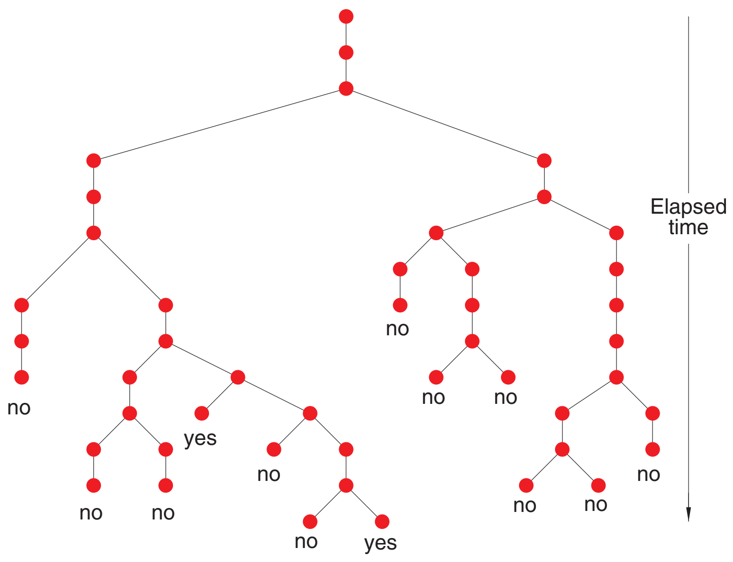

Figure 5. Example of a nondeterministic algorithm's execution history.

Nondeterministic algorithms

The definition of the second complexity class involves the concept of a *nondeterministic algorithm*, which is like an ordinary algorithm, except that it might use one additional, very powerful instruction:[10]

```
goto both label 1, label 2
```

This instruction splits the computation into two parallel processes, one continuing from each of the instructions, indicated by "label 1" and "label 2." By encountering more such instructions, the computation branches like a tree into several parallel computations that potentially can grow as an exponential function of the time elapsed (see Figure 5). A nondeterministic algorithm can perform an exponential number of computations in polynomial time.

In the world of conventional computers, nondeterministic algorithms are a theoretical concept only, but this could change in quantum computing. We need the concept of nondeterminism to define the class $\mathcal{NP}$ of "nondeterministic polynomial" problems:

Definition: A decision problem P is in the class $\mathcal{NP}$ if and only if a nondeterministic algorithm can solve it in polynomial time.

Solubility by a nondeterministic algorithm means this: All branches of the computation will stop, returning either "yes" or "no." We say that the overall algorithm returns "yes" if any of its branches returns "yes." The answer is "no" if none of the branches reports "yes." We say that a nondeterministic algorithm solves a decision problem in polynomial time if the number of steps used by the first of the branches to report "yes" is bounded by a polynomial in the problem's size.

We require polynomial solubility only for a decision problem's "yes" instances. This asymmetry between "yes" and "no" reflects the asymmetry between the "there is" and "for all" quantifiers in decision problems. A graph G is a "yes" instance of the Hamiltonian Cycle if there is at least one Hamiltonian cycle in G. For a "no" instance, all cycles in G must be non-Hamiltonian.

Conventional (deterministic) algorithms are special cases of nondeterministic algorithms (those nondeterministic algorithms that do not use the **goto both** instruction). It follows immediately that $\mathcal{P} \subseteq \mathcal{NP}$.

All decision problems discussed thus far have been members of $\mathcal{NP}$. Here's a nondeterministic polynomial algorithm for SAT:

```
Satisfiability ($F$)
**Input**: Boolean formula $F(x_1, ..., x_n)$
**Output**: 'yes' if $F$ is satisfiable, 'no' otherwise
**begin**
    **for** $i = 1$ **to** n
        **goto both** label 1, label 2
            label 1: $x_i$ = true; **continue**
            label 2: $x_i$ = false; **continue**
    **end**
    **if** $F(x_1, ..., x_n)$ = true  **then return** 'yes'
                                   **else return** 'no'
**end**
```

The **for** loop branches at each iteration: in one branch, x_i = true; in the other branch, x_i = false (the **continue** instruction starts the loop's next iteration). After executing the **for** loop, we have 2^n branches of computation—one for each of the possible assignments of n Boolean variables.

The power of nondeterministic algorithms is that they allow the exhaustive enumeration of an exponentially large number of candidate solutions in polynomial time. If the evaluation of each candidate solution (calculating $F(x_1, ..., x_n)$ in the SAT example) in turn can be done in polynomial time, the total nondeterministic algorithm is polynomial. For a problem from the class $\mathcal{NP}$, the sole source of intractability is the exponential size of the search space.

Succinct certificates

There is a second, equivalent definition of

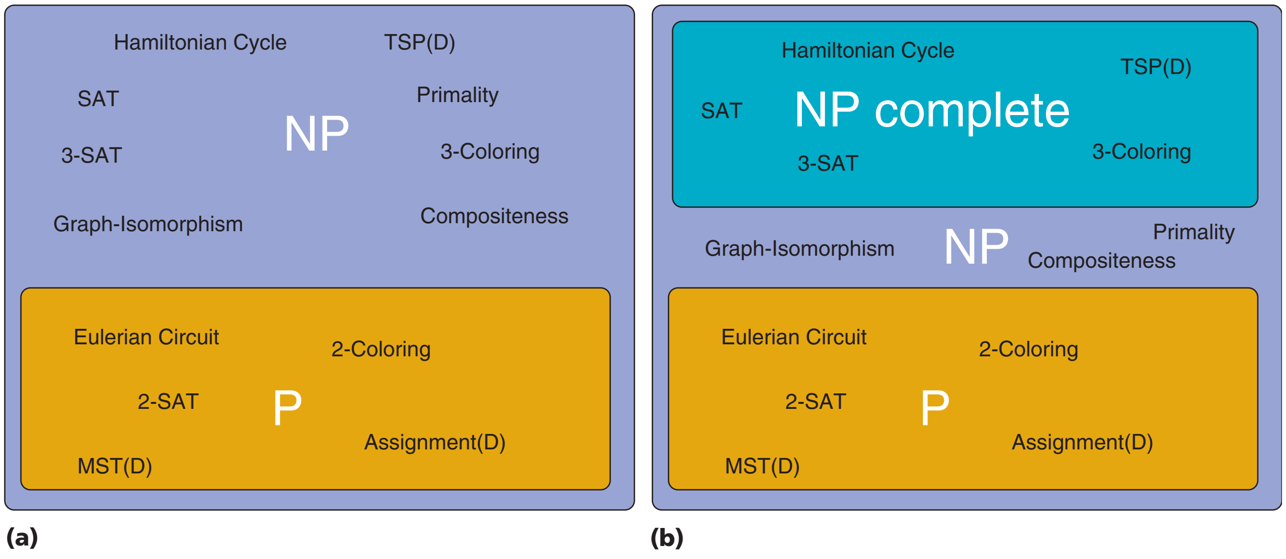

Figure 6. (a) A first map of complexity; (b) the map of complexity revisited. All problems indicated are defined within the text. Problems with a (D) are decision variants of optimization problems.

$\mathcal{NP}$ based on the notion of a *succinct certificate*. A certificate is a proof. If you claim that a graph G has a Hamiltonian Cycle, you can proof your claim by providing a Hamiltonian Cycle. Certificates for the Eulerian Circuit and k-Coloring are an Eulerian Circuit and a valid coloring, respectively. A certificate is succinct if its size is bounded by a polynomial in the problem size. The second definition then reads

Definition: A decision problem P is element of the class $\mathcal{NP}$ if and only if for every "yes" instance of P there exists a *succinct certificate* that can be verified in polynomial time.

The equivalence between both definitions can be proven easily.[10] The idea is that a succinct certificate can deterministically select the branch in a nondeterministic algorithm that leads to a "yes" output.

The definition based on nondeterministic algorithms reveals the key feature of the class $\mathcal{NP}$ more clearly, but the second definition is more useful for proving that a decision problem is in $\mathcal{NP}$. For example, consider

Compositeness: Given a positive integer N, are there integer numbers $p > 1$ and $q > 1$ such that $N = pq$?

A certificate of a "yes" instance N of Compositeness is a factorization $N = pq$. It is succinct because the number of bits in p and q is less than or equal to the number of bits in N, and it can be verified in quadratic time by multiplication. So, Compositeness $\in \mathcal{NP}$.

Most decision problems ask for the existence of an object with a given property, such as a cycle, which is Hamiltonian or a factorization with integer factors. In these cases, the desired object might serve as a succinct certificate. For some problems, this does not work, however. For example,

Primality: Given a positive integer N, is N prime?

Primality is the negation or complement of Compositeness: the "yes" instances of the former are the "no" instances of the latter and vice versa. A succinct certificate for Primality is by no means obvious. In fact, for many decision problems in $\mathcal{NP}$, no succinct certificate is known for the complement—that is, whether the complement is also in $\mathcal{NP}$ is not known. For Primality, however, there is a succinct certificate based on Fermat's Theorem.[11] Hence, Primality $\in \mathcal{NP}$.

A first map of complexity

Figure 6a summarizes what we have achieved so far. The class $\mathcal{NP}$ consists of all decision problems whose sole source of difficulty is the size of the search space, which grows exponentially with the problem size. These problems are intractable unless some mathematical insight provides us with a polynomial shortcut to avoid an exhaustive search. Such an insight promotes a problem into the class $\mathcal{P}$ of polynomially soluble problems.

The class $\mathcal{NP}$ not only contains many problems with important applications but also represents a real challenge: all problems in $\mathcal{NP}$ still have a chance to be in $\mathcal{P}$. A proof of nonexistence of a polynomial algorithm for a single problem from $\mathcal{NP}$ would establish that $\mathcal{P} \neq \mathcal{NP}$. As long as such a proof is missing,

$$P \overset{?}{=} NP \tag{7}$$

represents the most famous open conjecture in theoretical computer science.

$\mathcal{NP}$ completeness

So far, all the intractable problems seem to be isolated islands in the map of complexity. In fact, they are tightly connected by a device called *polynomal reduction*, which lets us map the computational complexity of one problem to the complexity of another. This finally leads to the surprising result that there are some intractable problems that are equivalent to all other problems. Each of these so called $\mathcal{NP}$-complete problems embodies the secret of intractability, since a polynomial time algorithm for one of them would immediately imply the tractability of all problems in $\mathcal{NP}$.

Polynomial reductions. The computational complexities of two problems, P_1 and P_2, can be related to each other by constructing an algorithm for P_1 that uses an algorithm for P_2 as a "subroutine." Consider the following algorithm that relates Hamiltonian Cycle to TSP(D):

```
Hamiltonian Cycle ($G$)
**Input**: Graph $G = (V, E)$
**Output**: 'yes' if $G$ contains a Hamiltonian cycle, 'no' otherwise
(1)  begin
(2)    $n := |V|$
(3)      for $i = 1$ to n
(4)        for $j = 1$ to n
(5)          if $(v_i, v_j) \in E$  then $d_{ij} := 1$
(6)                                 else $d_{ij} := 2$
(7)      if TSP-decision ($d, B := n$) = 'yes'
                              then return 'yes'
(8)                           else return 'no'
(9)  end
```

This algorithm solves the Hamiltonian Cycle by solving an appropriate instance of the TSP(D). In the **for** loops (lines 3 through 5), a distance matrix d is set up with entries $d_{ij} = 1$ if there is an edge (v_i, v_j) in G—otherwise, $d_{ij} = 2$. A Hamiltonian Cycle in G is a valid tour in the corresponding TSP with all intercity distances having length 1—that is, with total length n. Conversely, suppose that the TSP has a tour of length n. Because the intercity distances are either 1 or 2, and a tour sums up n such distances, a total length n implies that each pair of successively visited cities must have distance 1—that is, the tour follows existing edges in G and corresponds to a Hamiltonian Cycle. So, the call to a subroutine that solves TSP(D) (line 7) yields a solution to the Hamiltonian Cycle.

How does this algorithm relate the computational complexity of the Hamiltonian Cycle to that of the TSP(D)? This is a polynomial algorithm if the call to the TSP(D) solver is considered an elementary operation. If someone comes up with a polynomial algorithm for TSP(D), we will instantly have a polynomial algorithm for the Hamiltonian Cycle. We say that the Hamiltonian Cycle is *polynomially reducible* to TSP(D) and write

$$\text{Hamiltonian Cycle} \leq \text{TSP(D)}. \tag{8}$$

In many books, polynomial reducibility is denoted by $\propto$ instead of $\leq$. We use $\leq$ because this notation stresses an important consequence of polynomial reducibility:[12] the existence of a polynomial reduction from P_1 to P_2 excludes the possibility that we can solve P_2 in polynomial time but not P_1. Hence, we can read $P_1 \leq P_2$ as P_1 *cannot be harder than* P_2. Here is the (informal) definition:

Definition: We say a problem P_1 is polynomially reducible to a problem P_2 and write $P_1 \leq P_2$ if a polynomial algorithm exists for P_1 provided there is a polynomial algorithm for P_2.

NP-complete problems. Here are some other polynomial reductions that we can verify, similar to Equation 8 (the corresponding reduction algorithms appear elsewhere[13]):

$$\begin{array}{lcl} \text{SAT} & \leq & \text{3-SAT} \\ \text{3-SAT} & \leq & \text{3-Coloring} \\ \text{3-Coloring} & \leq & \text{Hamiltonian Cycle} \end{array} \tag{9}$$

Polynomial reducibility is transitive: $P_1 \leq P_2$ and $P_2 \leq P_3$ imply $P_1 \leq P_3$. From transitivity and Equations 8 and 9, it follows that each SAT, 3-SAT, 3-Coloring, and Hamiltonian Cycle reduces to TSP(D)—that is, a polynomial algorithm for TSP(D) implies a polynomial algorithm for all these problems. This is amazing, but it's only the beginning. Stephen Cook[14] revealed polynomial reducibility's true scope in 1971 when he proved that

Theorem: All problems in $\mathcal{NP}$ are polynomially reducible to SAT,

$$\forall P \in \mathcal{NP}: P \le \text{SAT}. \qquad (10)$$

This theorem means that

- No problem in $\mathcal{NP}$ is harder than SAT, or SAT is among the hardest problems in $\mathcal{NP}$.
- A polynomial algorithm for SAT would imply a polynomial algorithm for every problem in $\mathcal{NP}$—that is, it would imply $\mathcal{P} = \mathcal{NP}$.

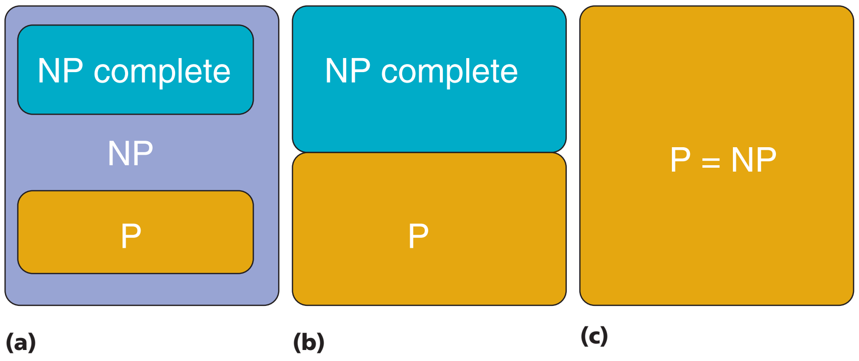

Figure 7. Three tentative maps of $\mathcal{NP}$. We can rule out map B; map A is likely (but not surely) the correct map. The discovery of a polynomial algorithm for any of the $\mathcal{NP}$-complete problems would turn C into the correct map.

It seems as if SAT is very special, but according to transitivity and Equations 8 and 9, 3-SAT, 3-Coloring, the Hamiltonian Cycle, or the TSP(D) can replace it. These problems form a new complexity class:

Definition: A problem P is called $\mathcal{NP}$ complete if $P \in \mathcal{NP}$ and $Q \le P$ for all $Q \in \mathcal{NP}$.

The class of $\mathcal{NP}$-complete problems collects the hardest problems in $\mathcal{NP}$. If any of them has an efficient algorithm, then every problem in $\mathcal{NP}$ can be solved efficiently—thus, $\mathcal{P} = \mathcal{NP}$. This is extremely unlikely, however, considered the futile efforts of many brilliant people to find polynomial algorithms for problems such as the Hamiltonian Cycle or TSP(D).

The map of $\mathcal{NP}$. Since Cook developed his theorem, many problems have been shown to be $\mathcal{NP}$ complete (see www.nada.kth.se/~viggo/wwwcompendium for a comprehensive, up-to-date list of hundreds of $\mathcal{NP}$-complete problems). Thus, the map of $\mathcal{NP}$ presented in Figure 6a needs some redesign (see Figure 6b). It turns out that all the intractable problems we have discussed so far are $\mathcal{NP}$ complete—except Compositeness and Primality. For both problems, neither a polynomial algorithm is known nor a polynomial reduction that classifies them as $\mathcal{NP}$ complete. Another $\mathcal{NP}$ problem that resists classification in either $\mathcal{P}$ or $\mathcal{NP}$ is

Graph Isomorphism: Given two graphs $G = (V, E)$ and $G'(V, E')$ on the same set of nodes, are G and G' isomorphic—that is, is there a permutation π of V such that $G' = \pi(G)$, where $\pi(G)$ denotes the graph $(V, \{[\pi(u), \pi(v)] : [u, v] \in E\})$?

There are more problems in $\mathcal{NP}$ that resist classification in $\mathcal{P}$ or $\mathcal{NP}$, but none of these problems has been proven not to belong to $\mathcal{P}$ or $\mathcal{NP}$. What has been proven is

Theorem: If $\mathcal{P} \ne \mathcal{NP}$, then $\mathcal{NP}$ problems exist that are neither in $\mathcal{P}$ nor $\mathcal{NP}$ complete.

This theorem[15] rules out one of three tentative maps of $\mathcal{NP}$ (see Figure 7).

Beyond $\mathcal{NP}$

The notions $\mathcal{NP}$ and $\mathcal{NP}$ complete strictly apply only to decision problems ("Is there a solution?"). The ideas of this approach can be generalized to optimization problems ("What is the best solution?") and counting problems ("How many solutions are there?").

Optimization problems. How does the classification of decision problems relate to optimization problems? The general instance of an optimization problem is a pair (F, c), where F is the set of feasible solutions and c is a cost function $c: F \to \mathbb{R}$. Here I consider only *combinatorial optimization* where the set F is countable. A combinatorial optimization problem P comes in three different flavors:

1. The optimization problem $P(O)$: Find the *feasible solution* $f^* \in F$ that minimizes the cost function.
2. The evaluation problem $P(E)$: Find the *cost* $c^* = c(f^*)$ of the minimum solution.

3. The decision problem $P(D)$: Given a bound $B \in \mathbb{R}$, is there a feasible solution $f \in F$ such that $c(f) \leq B$?

Under the assumption that we can evaluate the cost function c in polynomial time, it is straightforward to write down polynomial reductions that establish

$$P(D) \leq P(E) \leq P(O). \qquad (11)$$

If an optimization problem's decision variant is $\mathcal{NP}$ complete, there is no efficient algorithm for the optimization problem at all—unless $\mathcal{P} = \mathcal{NP}$. An optimization problem such as the TSP, whose decision variant is $\mathcal{NP}$ complete, is denoted $\mathcal{NP}$ hard.

Does a polynomial algorithm for a decision problem imply a polynomial algorithm for the optimization or evaluation variant? For that, we need to prove the reversal of Equation 11:

$$P(O) \leq P(E) \leq P(D). \qquad (12)$$

$P(E) \leq P(D)$ can be shown to hold if the optimum solution's cost is an integer with logarithm bounded by a polynomial in the input's size. The corresponding polynomial reduction evaluates the optimal cost c^* by asking, "Is $c^* \leq$ B?" for a sequence of values B that approaches c^* (similar to the bisection method for finding a function's zeroes).

There is no general method to prove $P(O) \leq P(E)$, but a strategy that often works for the TSP is this: Let c^* be the known solution of TSP(E). Replace an arbitrary entry d_{ij} of the distance matrix with a value $c > c^*$ and solve TSP(E) with this modified distance matrix. If this modification doesn't affect the tour's optimum length, the link ij does not belong to the optimal tour. Repeating this procedure for different links, we can reconstruct the optimum tour with a polynomial number of calls to a TSP(E) solver; hence, TSP(O) ≤ TSP(E).

Counting problems. So far, we have studied two related styles of problems: Decision problems ask whether a desired solution exists, and optimization problems require that a solution be produced. A third important and fundamentally different kind of problem asks how many solutions exist. The *counting* variant of SAT reads

#SAT: Given a Boolean expression, compute the number of different truth assignments that satisfy it.

Similarly, #Hamiltonian Cycle asks for the number of different Hamiltonian Cycles in a given graph, #TSP for the number of different tours with length $\leq B$, and so on.

Definition: A counting problem $\#P$ is a pair (F, d), where F is the set of all feasible solutions, and d is a decision function $d: F \to \{0,1\}$. The output of $\#P$ is the number of $f \in F$ with $d(f) = 1$. The class $\#\mathcal{P}$ (pronounced "number P") consists of all counting problems associated with a decision function d that can be evaluated in polynomial time.

Like the class $\mathcal{NP}$, $\#\mathcal{P}$ collects all problems whose sole source of intractability is the number of feasible solutions. A polynomial algorithm for a counting problem $\#P$ implies a polynomial algorithm for the associated decision problem P: $P \leq \#P$. Hence, it is unlikely that #SAT can be solved efficiently. In fact, we can define polynomial reducibility for counting problems and prove that all problems in $\#\mathcal{P}$ reduce polynomially to #SAT:[6]

Theorem: #SAT is $\#\mathcal{P}$ complete.

As you might have guessed, #Hamiltonian Cycle and #TSP are also $\#\mathcal{P}$ complete. Despite the similarity between $\mathcal{NP}$ and $\#\mathcal{P}$, counting problems are inherently harder than decision problems. This is documented by those $\#\mathcal{P}$-complete problems for which the corresponding decision problem can be solved in polynomial time—the classical example being the problem of calculating a matrix's permanent.[16]

Computational complexity and physics

The relationship between computation complexity and physics should offer new insights, some of which are discussed here. For example, some knowledge of computational complexity helps us understand the promises and limitations of quantum computers. In addition, we can seemingly transfer the notion of tractable and intractable problems to the problem of *analytical solubility* of models from statistical physics, explaining to some extent why, for example, the Ising model is exactly soluble in 2D but not 3D.

Another interesting link between computation complexity and physics is that statistical mechanics offer means for the general probabilistic analysis of computational problems. In statistical mechanics, we typically formulate an optimization problem as a spin glass and analyze the latter's low temperature properties. This "physical" approach often yields results that go beyond the results obtained by traditional methods. Another surprising observation is the fact that random instances of intractable problems can be solved in polynomial time. To observe the exponential running time, the parameters of the random ensemble must be adjusted carefully. This scenario corresponds to a phase transition in physical systems and is therefore best studied within the framework of statistical mechanics.

Quantum parallelism

There is some theoretical evidence that computers using quantum systems as computational devices are more powerful than computers based on classical devices. The hope is that problems that are intractable on a classical computer become tractable when put on a quantum computer. Results such as Peter Shor's celebrated quantum algorithm for factorization nurture this hope, but a real breakthrough is still missing.

In a seminal paper, Richard Feynman pointed out that a system of n quantum particles is exponentially hard to simulate on a classical computer.[17] The idea of quantum computing is to reverse this situation and simulate a classically hard (for example, exponential) problem in polynomial time on a computer made of quantum devices.

A quantum computer processes *qubits*—quantum two-state systems $|0\rangle$, $|1\rangle$. A quantum computer's key feature is that its registers can hold and process linear superpositions of all 2^n product states of n qubits, such as

$$\frac{1}{\sqrt{2^n}} \sum_{i_1, i_2, \ldots, i_n = 0}^{1} |i_1 i_2 \ldots i_n\rangle \ . \tag{13}$$

Using this feature, constructing a quantum computer capable of computing any function $f(x_1, \ldots, x_n)$ of n Boolean variables simultaneously for all 2^n possible input values is not difficult—in theory, at least. This *quantum parallelism* resembles a nondeterministic algorithm with its **goto both** instruction and its exponentially branching execution tree. Is quantum parallelism the key to exponential computing power? The problem is how to extract the exponential information out of a quantum computer. When we defined nondeterministic solubility, we didn't care about how to spot the single "yes" among the 2^n "no" answers. This works fine for a theoretical concept, but for a practical computer, reading the output really matters.

To gain the advantage of exponential parallelism, we must combine it with another quantum feature known as *interference*. The goal is to arrange the computation such that constructive interference amplifies desired result and destructive interference cancels the rest. Because of the importance of interference phenomena, it is not surprising that calculating the Fourier transform was the first problem that underwent an exponential speedup: from $\mathcal{O}(n \log n)$ on a classical to $\mathcal{O}(\log^2 n)$ on a quantum computer. This speedup was the seed for the most important quantum algorithm known today: Shor's algorithm to factor an integer in polynomial time.[18]

Although Shor's algorithm has some consequences for public key cryptography, it does not shatter the world of $\mathcal{NP}$: remember that Compositeness is in $\mathcal{NP}$, but it is not $\mathcal{NP}$ complete. Hence, the holy grail of quantum computing has yet to be discovered—a polynomial time quantum algorithm for an $\mathcal{NP}$-complete problem.

Analytical solubility of Ising models

Some problems in statistical physics have been exactly solved, but the majority of problems have withstood the efforts of generations of mathematicians and physicists. Why are some problems analytically solvable, whereas other, often similar, problems are not? Relating this question to the algorithmic complexity of evaluating the partition function gives us no final answer but helps clarify the borderline that separates analytically tractable from intractable problems.

For example, consider the Ising spin glass on a general graph G.[19] Let $\sigma = (\sigma_1, \ldots, \sigma_N)$ be an assignment of Ising spins $\sigma_i = \pm 1$ (up or down). The energy of a configuration σ is

$$E(\sigma) = -\sum_{\langle i,j \rangle} J_{i,j} \sigma_i \sigma_j - H \sum_i \sigma_i \ , \tag{14}$$

where H is the external magnetic field, $J_{i,j}$ are the coupling constants, and the first summation is over all edges in G. The fundamental problem in statistical mechanics is to determine the partition function

$$Z_N(G) = \sum_{\sigma} e^{-\beta E(\sigma)} \ . \tag{15}$$

Evaluating the sum directly requires $\mathcal{O}(2^N)$ operations. The notion of *analytic solution* has no precise definition, but as a minimum requirement, we want to reduce this number from being exponential to being polynomial in N.

Consider the well-known transfer matrix so-

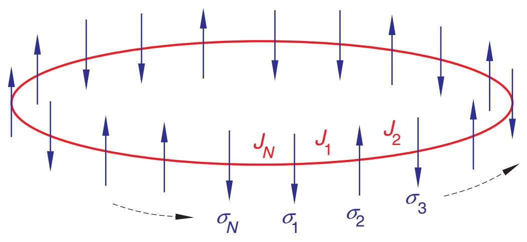

Figure 8. One-dimensional Ising spin glass with periodic boundary conditions. We can calculate this system's partition sum in polynomial time.

lution of the 1D Ising glass with periodic boundary conditions and coupling $\mathcal{J}_k$ between spins σ_k and σ_{k+1} (see Figure 8):

$$Z_N(\text{ring}) = \mathrm{Tr}\prod_{k=1}^{N}\begin{pmatrix} e^{\beta(\mathcal{J}_k+H)} & e^{-\beta\mathcal{J}_k} \\ e^{-\beta\mathcal{J}_k} & e^{\beta(\mathcal{J}_k-H)} \end{pmatrix}, \quad (16)$$

which can be evaluated in $\mathcal{O}(N)$ elementary operations. Because any exact solution must include all numbers $\mathcal{J}_k$, this is the best we can expect. In the homogeneous case $\mathcal{J}_k \equiv \mathcal{J}$, where we can calculate the product of transfer matrices,

$$Z_N = \lambda_+^N + \lambda_-^N \text{ with } \lambda_\pm = e^{\beta\mathcal{J}}\left[\cosh(\beta H) \pm \sqrt{\cosh^2(\beta H) - 2e^{-2\beta H}\sinh(2\beta H)}\right], \quad (17)$$

the evaluation complexity drops to $\mathcal{O}(\log N)$ using fast exponentiation.

Writing the partition sum as

$$Z(G) = \sum_{E_k} n(E_k)e^{-\beta E_k}, \quad (18)$$

where the sum is over all possible energy values, it becomes obvious that calculating Z is closely related to the $\#\mathcal{P}$ problem of determining $n(E_k)$. For general finite graphs G, this problem has proven to be $\#\mathcal{P}$ complete,[20] so there is no hope of finding an analytical solution (even in the weak sense above). This situation hardly improves if we restrict the graphs to the more "physical" crystal lattices: computing the partition function for a finite sublattice of a *nonplanar crystal lattice* is $\#\mathcal{P}$ complete.[21] This includes every crystal lattice in $d > 2$, the $d = 2$ model with next-nearest neighbor interactions, two coupled $d = 2$ models, and so forth. It also includes all models with $d \geq 2$ and external field H, because these can be transformed into zero-field models on an augmented graph $\hat{G}$ (which is nonplanar unless the underlying lattice graph G is 1D). Constructing $\hat{G}$ from G is easy—just adjoin an additional vertex (spin) σ_0 to G and let the additional edges $\sigma_0\sigma_i$ have constant interaction energy $\mathcal{J}_{0,i} = H$. The partition function of the augmented system reads

$$Z(\hat{G}) = \sum_{\sigma} e^{-\beta[-\sum \mathcal{J}_{i,j}\sigma_i\sigma_j]}\left(e^{-\beta H\sum\sigma_i} + e^{\beta H\sum\sigma_i}\right), \quad (19)$$

where the additional factor comes from the new spin $\sigma_0 = \pm 1$. From this expression, it is easy to see that $Z(\hat{G})$ equals two times the partition function $Z(G)$ in field H.

But where are the soluble Ising models? It has been proven that we can evaluate the partition function of Ising systems on *planar crystal lattices* in polynomial time.[22] This includes the celebrated Onsager solution of the square ferromagnet[23] as well as the 1D example just presented. It turns out that we can calculate the Ising model's partition sum in polynomial time for all graphs of fixed genus g.[24,25] A graph has genus g if it can be drawn on an orientable surface of genus g (a sphere with g "handles" attached to it) without crossing the edges. Planar graphs have genus 0, toroidal graphs have genus 1, and so on. For the crystal lattices in $d > 2$, the genus increases monotonically with the number of spins—that is, it is not fixed.[24]

The mechanism for proving tractability or intractability is the same in statistical mechanics as it is in computational complexity: polynomial reduction. Francisco Barahona, for example, applies a reduction of the $\mathcal{NP}$-hard problem Max Cut to the Ising spin glass in 3d to proof the latter's $\mathcal{NP}$ hardness.[22] A reduction of the planar Ising model to Minimum Weight Matching, on the other hand, proofs the tractability of the former because we can solve Minimum Weight Matching in polynomial time.

In our classification of spin systems, the nature of the couplings is not significant. A frustrated, glassy system with random couplings $\mathcal{J}_{i,j}$ of both signs is in the same class as the homogeneous ferromagnet with $\mathcal{J}_{i,j} \equiv \mathcal{J} > 0$ as long as the underlying graph G is the same. In the 1D example, we did not discriminate these cases: they are both polynomial. This situation changes, of course, if we consider the ground states rather than the complete partition function. Here, the nature of the couplings matters a lot: finding the ground states in pure ferromagnetic systems is trivial on all lattices, whereas it is $\mathcal{NP}$ hard for glassy systems with positive and negative couplings on nonplanar lattices.[22]

We can classify many other problems arising in statistical physics according to the computational complexity to evaluate their partition function.[26] We can also evaluate all the problems known to have an exact solution[27] in polynomial time. Problems that are #$\mathcal{P}$ complete, however, are unlikely to be exactly solvable. Anyone looking for an exact solution of such a problem should keep in mind that he or she is simultaneously attacking TSP, Hamiltonian Cycle, SAT, and all the other $\mathcal{NP}$-hard problems. In statistical mechanics, the focus is on results for the thermodynamic limit $N \to \infty$ rather than for finite systems, however. It is not clear how much of the "hardness" survives in this limit.

Probabilistic analysis of combinatorial problems

We can formally consider problems from combinatorial optimization as models in statistical mechanics. The cost function is renamed Hamiltonian, random instances are samples of quenched disorder, and the formal model's ground states correspond to the solutions of the optimization problems. In this way, methods developed in the framework of statistical mechanics of disordered systems become powerful tools for the probabilistic analysis of combinatorial problems.[28]

Researchers have applied statistical mechanics methods, for example, to the TSP,[29,30] Graph Partitioning,[31] and k-SAT.[32] A particularly nice example of this approach is the analysis of Assignment (also called Bipartite Matching): Given an $N \times N$ matrix with nonnegative entries $a_{i,j} \geq 0$, find

$$E_N^* = \min_{\sigma} \sum_{i=1}^{N} a_{i,\sigma(i)}, \qquad (20)$$

where the minimum is taken over all permutations σ of $(1, ..., N)$.

A probabilistic analysis aims at calculating average properties for an ensemble of random instances, the canonical ensemble being random numbers $a_{i,j}$ drawn independently from a common probability density $\rho(a)$. Using the replica method from statistical physics, Marc Mézard and Giorgio Parisi[33] found (among other things)

$$\lim_{N\to\infty} \left\langle E_N^* \right\rangle = \frac{\pi^2}{6}, \qquad (21)$$

where $\langle \bullet \rangle$ denotes averaging over the $a_{i,j}$. David Aldous recently presented[34] a rigorous proof of Equation 21, which represents one of the rare cases where rigorous methods have confirmed a replica result.

Some years after the replica solution, Parisi recognized that for exponentially distributed matrix elements ($\rho(a) = e^{-a}$), the average optima for $N = 1$ and $N = 2$ are

$$\left\langle E_1^* \right\rangle = 1 \qquad \left\langle E_2^* \right\rangle = 1 + \frac{1}{2^2}. \qquad (22)$$

From this and the fact that we can write the replica result for $N\to\infty$ as

$$\frac{\pi^2}{6} = \sum_{k=1}^{\infty} \frac{1}{k^2} \qquad (23)$$

he conjectured[35] that the average optimum for finite systems is

$$\left\langle E_N^* \right\rangle = \sum_{k=1}^{N} \frac{1}{k^2}. \qquad (24)$$

Parisi's conjecture is supported by numerical simulations, but no formal proof has been found despite some efforts.[36]

Equations 22 and 24 only hold for $\rho(a) = e^{-a}$, whereas Equation 21 is valid for all densities with $\rho(a \to 0) = 1$. For the uniform density on [0,1], the first terms are

$$\left\langle E_1^* \right\rangle = \frac{1}{2} \qquad \left\langle E_2^* \right\rangle = \frac{23}{30}. \qquad (25)$$

(If you can you guess the expression for general, finite N in this case, please send me an email.)

Sometimes a statistical mechanics analysis not only yields exact analytical results but also reveals features that are important to design and understand algorithms. A recent example is the analysis of the Davis-Putnam algorithm for SAT.[37] Another example is given by the number partitioning problem (NPP), an $\mathcal{NP}$-hard optimization problem.[38] "Physical" reasoning has led to the conjecture that for this problem, no heuristic algorithm can outperform simple random search.[39]

Phase transitions in computational complexity

The theory of computational complexity is based entirely on worst-case analysis. An algorithm could require exponential time on pathological instances only. A famous example is the simplex method for linear programming. Despite its exponential worst-case complexity, it is used in many applications to solve really large problems. Apparently the instances that trigger exponential running time do not appear under practical conditions.

Linear programming is in $\mathcal{P}$ thanks to the Ellipsoid algorithm,[5] but similar scenarios are observed for $\mathcal{NP}$-hard problems. Take 3-SAT, for example. When generating random instances with N variables and M clauses and feeding them to a smart but exhaustive algorithm, we observe polynomial running time unless the ratio M/N is carefully adjusted. If M/N is too low, the problem is *underconstrained*—that is, it has many satisfying solutions, and a clever algorithm will find one of these quickly. If M/N is too large, the problem is *overconstrained*—that is, it has many contradictory constraints, which, again, a clever algorithm will discover quickly.[40] The real hard instances are generated for ratios $\alpha = M/N$ close to a critical value α_c.[41]

The transition from underconstrained to overconstrained formulas in 3-SAT bears the hallmarks of a phase transition in physical systems. The control parameter is α; the order parameter is the probability of the formula being satisfiable. Similar phase transitions do occur in various other decision or optimization problems, and mathematical methods from statistical mechanics have successfully been used for their analysis.

Reading up to this point hopefully has convinced you that there are some interesting links between physics and the theory of computational complexity. In fact, mathematicians, computer scientists, and physicists have just started interdisciplinary work in this field. Tools and notions from statistical mechanics might shed more light on the typical case complexity of problems and might help to improve heuristic algorithms. Quantum computing may even turn intractable problems into tractable ones some day. A polynomial time algorithm for an $\mathcal{NP}$-complete problem would be a real breakthrough—a theoretical breakthrough first of all, but in the far future, quantum computers might be available as hardware. If this really happens, computational complexity and physics will no longer be considered separate fields. CiSE

References

1. D. Coppersmith and S. Winograd, "Matrix Multiplication via Arithmetic Progressions," *J. Symbolic Computation*, vol. 9, no. 3, 1990, pp. 251–280.
2. T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms: The MIT Electrical Engineering and Computer Science Series,* MIT Press, Cambridge, Mass., 1990.
3. B. Bollobás, *Modern Graph Theory*, Graduate Texts in Mathematics, vol. 184, Springer-Verlag, Berlin, 1998.
4. G. Reinelt, "The Traveling Salesman," *Computational Solutions for TSP Applications*, Lecture Notes in Computer Science, vol. 840, Springer-Verlag, Berlin, 1994.
5. C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice-Hall, Englewood Cliffs, N.J., 1982.
6. L. Euler, "Solutio Problematis Ad Geometrian Situs Pertinentis," *Commetarii Academiae Scientiarum Imperialis Petropolitanae,* vol. 8, 1736, pp. 128–140.
7. H.R. Lewis and C.H. Papadimitriou, "The Efficiency of Algorithms," *Scientific American*, vol. 109, no. 1, Jan. 1978, pp. 96–109.
8. V. Kumar, "Algorithms for Constraint Satisfaction Problems: A Survey," *AI Magazine*, vol. 13, no. 1, 1992, pp. 32–44; ftp://ftp.cs.umn.edu/dept/users/kumar/csp-aimagazine.ps.
9. B. Aspvall, M.F. Plass, and R.E. Tarjan, "A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas," *Information Processing Letters*, vol. 8, no. 3, 1979, pp. 121–123.
10. D.S. Johnson and C.H. Papadimitriou, "Computational Complexity," *The Traveling Salesman Problem*, Lawler et al., eds., 1985, pp. 37–85.
11. V.R. Pratt, "Every Prime has a Succinct Certificate," *SIAM J. Computing*, vol. 4, no. 3, 1975, pp. 214–220.
12. G. Ausiello et al., *Complexity and Approximation*, Springer-Verlag, Berlin, 1999.
13. R.M. Karp, "Complexity of Computer Computations," *Reducibility Among Combinatorial Problems*, R.E. Miller and J.W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.
14. S. Cook, "The Complexity of Theorem Proving Procedures," *Proc. 3rd Ann. ACM Symp. Theory of Computing*, ACM Press, New York, 1971, pp. 151–158.
15. R.E. Ladner, "On the Structure of Polynomial Time Reducibility," *J. ACM*, vol. 22, no. 1, 1975, pp. 155–171.
16. L.G. Valiant, "The Complexity of Computing the Permanent," *Theoretical Computer Science*, vol. 8, 1979, pp. 189–201.
17. R. Feynman, "Simulating Physics with Computers," *Int'l J. Theoretical Physics*, vol. 21, nos. 6 and 7, 1982, pp. 467–488.
18. P.W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM J. Computing*, vol. 26, no. 5, 1997, pp. 1484–1509.
19. B. Hayes, "Computing Science: The World in a Spin," *American Scientist*, vol. 88, no. 5, Sept./Oct. 2000, pp. 2384–388; www.amsci.org/amsci/issues/comsci00/compsci2000-09.html.
20. F. Jaeger, D.L. Vertigan, and D.J.A. Welsh, "On the Computational Complexity of the Jones and Tutte Polynomials," *Mathematical Proc. Cambridge Philosophical Soc.*, vol. 108, 1990, pp. 35–53.
21. S. Istrail, "Statistical Mechanics, Three-Dimensionality and NP-Completeness," *Proc. 31st ACM Ann. Symp. Theory of Computing* (STOC 2000), ACM Press, New York, 2000, pp. 87–96.
22. F. Barahona, "On the Computational Complexity of Ising Spin Glass Models," *J. Physics A: Mathematical and General*, vol. 15, vol. 10, 1982, pp. 3241–3253.
23. L. Onsager, "Crystal Statistics I: A Two-Dimensional Model with an Order-Disorder Transition," *Physical Rev.*, vol. 65, 1944, pp. 117–149.
24. T. Regge and R. Zecchina, "Combinatorial and Topological Approach to the 3D Ising Model," *J. Phys. A*, vol. 33, 2000, pp. 741–761.
25. A. Galluccio, Martin Loebl, and Jan Vondrák, "New Algorithm for the Ising Problem: Partition Function for Finite Lattice Graphs," *Physical Rev. Letters*, vol. 84, no. 26, 2000, pp. 5924–5927.
26. D.J.A. Welsh, "The Computational Complexity of Some Classical Problems from Statistical Physics," *Disorder in Physical Sys-*

tems, G.R. Grimmett and D.J.A. Welsh, eds., Clarendon Press, Oxford, 1990, pp. 307–321.

27. R.J. Baxter, *Exactly Solved Models in Statistical Mechanics*, Academic Press, San Diego, 1982.
28. M. Mézard, G. Parisi, and M.A. Virasoro, *Spin Glass Theory and Beyond*, World Scientific, Singapore, 1987.
29. M. Mézard and G. Parisi, "Mean-Field Equations for the Matching and the Traveling Salesman Problems," *Europhys. Letters*, vol. 2, no. 12, Dec. 1986, pp. 913–918.
30. W. Krauth and M. Mézard, "The Cavity Method and the Traveling-Salesman Problem," *Europhys. Lett.*, vol. 8, no. 3, 1989, pp. 213–218.
31. Y. Fu and P.W. Anderson, "Application of Statistical Mechanics to NP-Complete Problems in Combinatorial Optimization," *J. Physics A: Mathematical and General*, vol. 19, 1986, pp. 1605–1620.
32. R. Monasson and Riccardo Zecchina, "Statistical Mechanics of the Random k-Satisfiability Model," *Phys. Rev. E*, vol. 56, no. 2 Aug. 1997, pp. 1357–1370.
33. M. Mézard and G. Parisi, "Replicas and Optimization," *J. Physique Letters*, vol. 46, 1985, pp. L771–L778.
34. D. J. Aldous, "The ζ(2) Limit in the Random Assignment Problem," *Random Structures & Algorithms*, vol. 18, no. 4, July 2001, pp. 381–418.
35. G. Parisi, *A Conjecture on Random Bipartite Matching*, 1998; http://xxx.lanl.gov/PS_cache/cond-mat/pdf/9801/9801176.pdf.
36. D. Coppersmith and G. Sorkin, "Constructive Bounds and Exact Expectations for the Random Assignment Problem," *Random Structures and Algorithms*, vol. 15, no. 2, 1999, pp. 113–144.
37. S. Cocco and R. Monasson, "Statistical Physics Analysis of the Computational Complexity of Solving Random Satisfiability Problems using Backtrack Algorithms," *European Physics J. B*, vol. 22, 2001, pp. 505–531.
38. B. Hayes, "Computing Science: The Easiest Hard Problem," *American Scientist*, vol. 90, no. 2, Mar./Apr. 2002, pp. 113–117; www.amsci.org/amsci/issues/comsci02/compsci2002-03.html.
39. S. Mertens, "Random Costs in Combinatorial Optimization," *Physical Rev. Letters*, vol. 84, no. 7, Feb. 2000, pp. 1347–1350.
40. B. Hayes, "Computing Science: Can't Get No Satisfaction," *American Scientist*, vol. 85, no. 2, Mar./Apr. 1997, pp. 108–112; www.amsci.org/amsci/issues/Comsci97/compsci9703.html.
41. R. Monasson et al., "Determining Computational Complexity from Characteristic 'Phase Transitions,'" *Nature*, vol. 400, July 1999, pp. 133–137.

Stephan Mertens is a teaching and research assistant at the Institute for Theoretical Physics, University of Magdeburg. His fields of research are statistical mechanics of hard optimization problems, phase transitions in computational complexity, and parallel algorithms. He runs the Institute's computing machinery, including a 156-node Beowulf cluster. He received his PhD in theoretical physics from Göttingen University. Contact him at Otto-von-Guericke Univ., Inst. f. Theor. Physik, Otto-von-Guericke Univ., PF 4120, D-39016 Magdeburg, Germany; stephan.mertens@physik.uni-magdeburg.de.