AMERICAN Scientist

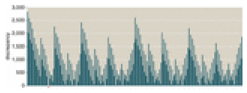
COMPUTING SCIENCE

The Easiest Hard Problem

Brian Hayes

One of the cherished customs of childhood is choosing up sides for a ball game. Where I grew up, we did it this way: The two chief bullies of the neighborhood would appoint themselves captains of the opposing teams, and then they would take turns picking other players. On each round, a captain would choose the most capable (or, toward the end, the least inept) player from the pool of remaining candidates, until everyone present had been assigned to one side or the other. The aim of this ritual was to produce two evenly matched teams and, along the way, to remind each of us of our precise ranking in the neighborhood pecking order. It usually worked.

None of us in those days—not the hopefuls waiting for our name to be called, and certainly not the two thick-necked team leaders—recognized that our scheme for choosing sides implements a greedy heuristic for the balanced number partitioning problem. And we had no idea that this problem is NP-complete—that finding the optimum team rosters is certifiably hard. We just wanted to get on with the game.


+ enlarge image

And therein lies a paradox: If computer scientists find the partitioning problem so intractable, how come children the world over solve it every day? Are the kids *that* much smarter? Quite possibly they are. On the other hand, the success of playground algorithms for partitioning might be a clue that the task is not always as hard as that forbidding term "NP-complete" tends to suggest. As a matter of fact, finding a hard instance of this famously hard problem can be a hard problem—unless you know where to look. Some recent results, which make use of tools borrowed from physics and mathematics as well as computer science, have now shown exactly where the hard problems hide.

The organizers of sandlot ball games are not the only ones with an interest in efficient partitioning. Closely related problems arise in many other resource-allocation tasks. For example, scheduling a series of jobs on a dual-processor computer is a partitioning problem: Sorting the jobs into two sets with equal running time will balance the load on the processors. Another example is apportioning the miscellaneous assets of an estate between two heirs.

So What's the Problem?

Here is a slightly more formal statement of the partitioning problem. You are given a set of *n* positive integers, and you are asked to separate them into two subsets; you may put as many or as few numbers as you please in each of the subsets, but you must make the sums of the subsets as nearly equal as possible. Ideally, the two sums would be exactly the same, but this is feasible only if the sum of the entire set is even; in the event of an odd total, the best you can possibly do is to choose two subsets that differ by 1. Accordingly, a perfect partition is defined as any arrangement for which the "discrepancy"—the absolute value of the subset difference—is no greater than 1.

Try a small example. Here are 10 numbers—enough for two basketball teams—selected at random from the range between 1 and 10:

2 10 3 8 5 7 9 5 3 2

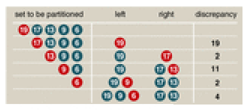
Can you find a perfect partition? In this instance it so happens there are 23 ways to divvy up the numbers into two groups with exactly equal sums (or 46 ways if you count mirror images as distinct partitions). Almost any reasonable method will converge on one of these perfect solutions. This is the answer I stumbled onto first:

(2 5 3 10 7) (2 5 3 9 8)

Both subsets sum to 27.

This example is in no way unusual. As a matter of fact, among all sets of 10 integers between 1 and 10, more than 99 percent have at least one perfect partition. (To be precise, of the 10 billion such sets, 9,989,770,790 can be perfectly partitioned. I know because I counted them—and it wasn't easy.)

Maybe larger sets are more challenging? With a list of 1,000 numbers between 1 and 10, working the problem by pencil-and-paper methods gets tedious, but a simple computer program makes quick work of it. A variation on the two-bullies algorithm does just fine. First sort the list of numbers according to magnitude, then go through them in descending order, assigning each number to whichever subset currently has the smaller sum. This is called a greedy algorithm, because it takes the largest numbers first.

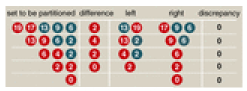

+ enlarge image

The greedy algorithm almost always finds a perfect partition for a list of a thousand random numbers no greater than 10. Indeed, the procedure works equally well on a set of 10,000 or 100,000 or a million numbers in the same range. The explanation of this success is not that the algorithm is a marvel of ingenuity. Lots of other methods do as well or better.

A particularly clever algorithm was described in 1982 by Narendra Karmarkar and Richard M. Karp, who were then both at the University of California, Berkeley. It is a "differencing" method: At each stage you choose two numbers from the set to be partitioned and replace them by the absolute value of their difference. This operation is equivalent to deciding that the two selected integers will go into different subsets, without making an immediate commitment about which numbers go where. The process continues until only one number remains in the list; this final value is the discrepancy of the partition. You can reconstruct the partition itself by working backward through the series of decisions. In the search for perfect partitions, the Karmarkar-Karp procedure succeeds even more often than the greedy algorithm.

At this point you may be ready to dismiss partitioning as just a wimpy problem, unworthy of the designation NP-complete. But try one more example. Here is another list of 10 random numbers, chosen not from the range 1 to 10 but rather from the range between 1 and 2^{10}, or 1,024:

771 121 281 854 885 734 486 1003 83 62


+ enlarge image

This set does have a perfect partition, but there is just one, and finding it takes a little persistence. The greedy algorithm does not succeed; it gets stuck on a partition with a discrepancy of 32. Karmarkar-Karp does slightly better, reducing the discrepancy to 26. But the only sure way to find the one perfect partition is to check all possible partitions, and there are 1,024 of them.

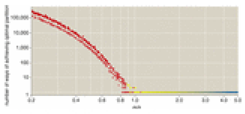
If this challenge is still not daunting enough, try 100 numbers ranging up to 2^{100}, or 1,000 numbers up to 2^{1000}. Unless you get very lucky, deciding whether such a set has a perfect partition will keep you busy for quite a few lifetimes.

Where the Hard Problems Are

To make sense of what's going on here, it's necessary first to be clear about what it means for a problem to be hard. Computer science has reached a rough consensus on this issue: Easy problems can be solved in "polynomial time," whereas hard problems require "exponential time." If you have a problem of size *x*, and you know an algorithm that can solve it in *x* steps or x^2 steps or even x^{50} steps, then the problem is officially easy; all of these expressions are polynomials in *x*. But if your best algorithm needs 2^x steps or x^x steps, you're in trouble. Such exponential functions grow faster than any polynomial for large enough values of *x*.

The easy, polynomial-time problems are said to lie in class P; hard problems are all those *not* in P. The notorious class NP consists of some rather special hard problems. As far as anyone knows, solving these problems requires exponential time. On the other hand, if you are given a proposed solution, you can check its correctness in polynomial time. (NP stands for "nondeterministic polynomial"—although admittedly that's not much help in understanding the concept.)

Where does number partitioning fit into this taxonomy? Both the greedy algorithm and Karmarkar-Karp have polynomial running time; they can partition a set of *n* numbers in less than n^2 steps. For purposes of classification, however, these algorithms simply don't count, because they're not guaranteed to find the right answer. The hierarchy of problem difficulty is based on a worst-case analysis, which disqualifies an algorithm if it fails on even one problem instance. The only known method that does pass the worst-case test is the brute-force approach of examining every possible partition. But this is an exponential-time algorithm: Each integer in the set can be assigned to either of the two subsets, so that there are 2^n partitions to be considered.



✚ enlarge image

Partitioning is a classic NP problem. If someone hands you a list of *n* numbers and asks, "Does this set have a perfect partition?" you can always find the answer by exhaustive search, but this can take an exponential amount of time. If you are given a proposed perfect partition, however, you can easily verify its correctness in polynomial time. All you need to do is add up the two subsets and compare the sums, which takes time proportional to *n*.

Indeed, partitioning is not just an ordinary member of the class NP; it is one of the elite NP problems designated NP-complete. What this means is that if someone discovered a polynomial-time algorithm for partitioning, it could be adapted to solve *all* NP problems in polynomial time. Each NP-complete problem is a skeleton key to the entire class NP.

Given these sterling credentials as a hard problem, it's all the more perplexing that partitioning often yields so readily to simple and unsophisticated methods such as the greedy algorithm. Does this problem have teeth, or is it just a sheep in wolf's clothing?

Boiling and Freezing Numbers

An answer to this question has emerged in the past few years from a campaign of research that spans at least three disciplines—physics, mathematics and computer science. It turns out that the spectrum of partitioning problems has both hard and easy regions, with a sharp boundary between them. On crossing that frontier, the problem undergoes a phase transition, analogous to the boiling or freezing of water.

The standard classification of computing problems as P or NP and so on assumes that difficulty increases as a function of problem size. In the case of number partitioning, two factors determine the size of a problem instance: how many numbers are in the set, and how big they are. Specifically, the size of an instance is the number of bits needed to represent it, and this depends both on the number of integers, *n*, and on the number of bits, *m*, in a typical integer. Thus a set of 100 integers in a range near 2^{100} has *n*=100 and *m*=100 and a problem size of 10,000 bits.

Given this measure of size, one might expect that partitioning problems would get harder as the product of *n* and *m* increases. This conclusion is not entirely wrong; algorithms do have to labor longer over bigger problems, if only to read the input. Yet, surprisingly, the product *nm* is not the best predictor of difficulty in number partitioning. Instead it's the ratio *m*/*n*.

Some simple reasoning about extreme cases takes the mystery out of this assertion. Suppose the ratio of *m* to *n* is very small, say with *m*=1 and *n*=1,000. The task, then, is to partition a set of 1,000 numbers, each of which can be represented by a single bit. This is trivially easy: A one-bit positive integer must be equal to 1, and so the input to the problem is a list of a thousand 1s. Finding a perfect partition is just a matter of counting. At the opposite extreme, consider the case of *m*=1,000 and *n*=2 (the smallest *n* that makes sense in a partitioning problem). Here the separation into subsets is easy—how many ways can you partition a set of two items?—but the likelihood of a perfect partition is extremely low. It's just the probability that two randomly selected 1,000-bit numbers will be equal.

Now it becomes clear why in my baby-boom neighborhood we so easily formed ourselves into well-matched teams. Among the dozen or more kids who would gather for a game, athletic abilities may have differed by a factor of 10, but surely not by a factor of 1,000. The parameter *m* is the base-2 logarithm of this range of talents, and so it was no greater than 3 or 4. Thus *m*/*n* was rather small—less than 1—and we had many acceptable solutions to choose from. Perhaps if we had had a young Michael Jordan or Mia Hamm in the neighborhood, I would take a different view of the number-partitioning problem today.

Antimagnetic Numbers

The ratio *m*/*n* divides the space of partitioning problems into two regions. Somewhere between them—between the fertile valley where solutions bloom everywhere and the stark desert where even one perfect partition is too much to expect—there must be a crossover region. There lies the phase transition.

The concept of a phase transition comes from physics, but it also has a long history of applications to mathematical objects. Forty years ago Paul Erdo?s and Alfred Rényi described phase transitions in the growth of random graphs (collections of vertices and connecting edges). By the 1980s, phase transitions had been observed in many combinatorial processes. The most thoroughly explored example is an NP-complete problem called satisfiability. A 1991 article by Peter Cheeseman, Bob Kanefsky and William M. Taylor, titled "Where the *Really* Hard Problems Are," conjectured that *all* NP problems have a phase transition and suggested that this is what distinguishes them from problems in P.

Meanwhile, in another paper with a provocative title ("The Use and Abuse of Statistical Mechanics in Computational Complexity"), Yaotian Fu of Washington University in St. Louis argued that number partitioning is an example of an NP problem *without* a phase transition. This assertion was disputed by Ian P. Gent of the University of Strathclyde and Toby Walsh of the University of York, who presented strong computational evidence for the existence of a phase transition. Their measurements suggested that the critical value of the *m*/*n* ratio, where easy problems give way to hard ones, is about 0.96.

Stephan Mertens, of Otto von Guericke Universität in Magdeburg, Germany, has now given a thoroughgoing analysis of number partitioning from a physicist's point of view. His survey paper (which has been my primary source in writing this article) appears in a special issue of *Theoretical Computer Science* devoted to phase transitions in combinatorial problems.

As a means of understanding the phase transition, Mertens sets up a correspondence between the number-partitioning problem and a model of a physical system. To see how this works, it helps to think of the partitioning process in a new context. Instead of unzipping a list of numbers into two separate lists, keep all the numbers in one place and multiply some of them by –1. The idea is to negate just the right selection of numbers so that the entire set sums to 0. Now comes the leap from mathematics to physics: The collection of positive and negative numbers is analogous to an array of atoms in a magnetic material, with the plus and minus signs representing up and down spins. Specifically, the system resembles an infinite-range antiferromagnet, where every atom can feel the influence of every other atom, and where the favored configuration has spins pointing in opposite directions.

This strategy for studying partitioning may seem slightly perverse. It takes a simply stated problem in combinatorics and turns it into a messy and rather obscure system in statistical mechanics. Why bother? The reason is that physics offers some powerful tools for predicting the behavior of such a system. In particular, the interplay of energy and entropy governs how the collection of spins can be expected to evolve toward a state of stable equilibrium. The energy in question comes from the interaction between atomic spins (or between positive and negative numbers); because the system is an antiferromagnet, the energy is minimized when the spin vectors are oppositely oriented (or when the subsets sum to zero). The entropy measures the number of ways of achieving the minimum-energy state; a system with a unique ground state (or just one perfect partition) has zero entropy. When there are many equivalent ways of minimizing the energy (or partitioning a set perfectly), the entropy is high.

The ratio *m*/*n* controls the state of this system. When *m* is much greater than *n*, the spins almost always have just one configuration of lowest energy. At the other pole, when *m* is much smaller than *n*, there are a multitude of zero-energy states, and the system can land in any one of them. Mertens showed that the transition between the two phases comes at *m*/*n*=1, at least in the limit of very large *n*. And he derived corrections for finite *n* that may explain why Gent and Walsh measured a slightly different transition point.

Finally, Mertens showed just how hard the hard phase is. Searching for the best partition in this region is equivalent to searching a randomly ordered list of random numbers for the smallest element of the list. Only an exhaustive traverse of the entire list can guarantee an exact result. What's worse, there are no really good heuristic methods; no shortcuts are inherently superior to blind, random sampling.

Yet this is not to say that heuristics for partitioning are totally worthless. On the contrary, the phase-transition model helps explain how the Karmarkar-Karp algorithm works. The differencing operation reduces the range of the numbers and so diminishes *m*/*n*, sliding the problem toward the easy phase. The algorithm can't be counted on to find the very best partition, but it's an effective way of avoiding the worst ones.

Back to Mathematics

The work of Mertens has answered most of the major questions about number partitioning, and yet it's not quite the end of the story. The methods of statistical mechanics were developed as tools for describing systems made up of vast numbers of component parts, such as the atoms of a macroscopic specimen of matter. When applied to number partitioning, the methods are strictly valid only for very large sets, where *m* and *n* both go to infinity (while maintaining a fixed ratio). Those who need to solve practical partitioning problems are generally interested in somewhat smaller values of *m* and *n*.

Mertens's results have another limitation as well. In calculating the distribution of optimal solutions, he had to adopt a simplifying approximation at one crucial step, assuming that certain energies in the spin system are random. The true distribution of those energies is harder to deduce, but the task has been undertaken by Christian Borgs and Jennifer T. Chayes of Microsoft Research and Boris Pittel of Ohio State University. They have reclaimed the problem from the realm of physics and brought it back to mathematics. Their paper giving detailed proofs runs to nearly 40 pages.

To their surprise, Borgs, Chayes and Pittel discovered that Mertens's random-energy approximation actually yields exact results in the limiting case of infinite *m* and *n*. Under these conditions the phase transition is perfectly sharp. Anywhere below the critical ratio *m*/*n*=1, the probability of a perfect partition is 1; above this threshold the probability is 0. Borgs, Chayes and Pittel also give a precise account of how the phase transition softens and broadens for smaller problems. When *n* is finite, the probability of a perfect partition varies smoothly between 0 or 1, with a "window" of finite width surrounding the critical ratio.

If my friends and I back on the ball field had known all this, would we have played better games? Probably not, but in retrospect I take satisfaction in the thought that our ritual for choosing teams was algorithmically well-founded.