

SIMONS FOUNDATION

Advancing Research in Basic Science and Mathematics

[Home](#) » [Mathematics and Physical Science](#) » Approximately Hard: The Unique Games Conjecture

Approximately Hard: The Unique Games Conjecture

A new conjecture has electrified computer scientists

by: **Erica Klarreich**

October 6, 2011

While great theorems are arguably the end goal of mathematics, it is often a great conjecture that does the most to advance understanding and create new theories. For no field has this been truer than computational complexity, the study of which computational problems are within reach of a standard computer. The famous P versus NP conjecture, still unsolved after 40 years, has done more than perhaps any other single mathematical assertion in modern times to set the ground rules and the very language of its field.



Subhash Khot

Formulated in 1971, the conjecture focused the attention of computer scientists on the contrast between two classes of problems: P, the class of problems a computer can solve efficiently, and NP, the class of problems for which a computer can check efficiently whether a proposed solution is correct, even if it doesn't have the capability to come up with that solution in the first place. Natural computation problems tend to fall into one of these two categories. The conjecture states—and most computer scientists believe—that these two classes are distinct, meaning that there are some problems in NP whose solutions are beyond the reach of any efficient computer algorithm.

Now, the field of computational complexity has been electrified by a new conjecture, which, while not as monumental as the P versus NP conjecture, in some sense picks up where that conjecture leaves off. If the 'Unique Games Conjecture' is correct, then for many of the problems people would most like to solve, it's hard not only to find an exact solution—finding even a good approximation of the solution is beyond the reach of a computer.

"The conjecture is an anchor to which we can connect an enormous number of approximation problems," says Avi Wigderson of the Institute of Advanced Study in Princeton. "It creates a theory."

Over the four decades since the original P versus NP conjecture was formulated, it has stubbornly resisted proof. But at the same time, computer scientists have amassed a wealth of evidence suggesting that it is correct. They have shown that thousands of practical problems—for example, figuring out the best design for a chip, the best airplane schedule, or the lowest-energy configuration of a protein—are NP-hard, meaning that an efficient

algorithm for any one of them would automatically produce an efficient solution to *all* the problems in NP (which would mean that $P = NP$). So far, no one has managed to produce an efficient algorithm for any one of these thousands of problems, and not for want of trying. If the conjecture is correct, no one ever will.

(For more on the notion of efficiency and the P versus NP conjecture, see the sidebar, *Efficient Solutions*.)

But suppose that instead of, say, a perfect chip design, you would be content with one that is simply pretty good. Over the past few decades, computer scientists have come up with good approximation algorithms for some such problems, and for other problems they have proved that good approximation algorithms do not exist. However, for many of the problems people are most interested in solving, computer scientists are in the dark about whether, and how well, they can be approximated.

Now, the Unique Games Conjecture (UGC) has given researchers a new hook on which to try to catch some of these slippery problems. Formulated in 2002 by Subhash Khot, currently at New York University, the UGC simply proposes that one particular problem about assigning colors to the nodes of a network is NP-hard to solve even approximately.

This conjecture may seem, on the face of it, like a pretty parochial statement (though possibly hard to prove, or even false). Over the past decade, however, computer scientists have found to their surprise that if the conjecture is true, then a host of problems—some of them seeming to bear little resemblance to the original Unique Games problem—are also hard to approximate. The findings have left researchers scratching their heads about just what is so special about this particular network-coloring problem.

“The conjecture seems to be the magic key that makes a lot of problems more simple and clean,” says Ryan O’Donnell, of Carnegie Mellon University in Pittsburgh. “It has changed the way we think about these problems.”

Like many great conjectures, the UGC has led to findings in a wide range of fields seemingly far removed from computational complexity, including the structure of foams, the geometry of different ways to measure distance, and even the merits of different voting systems.

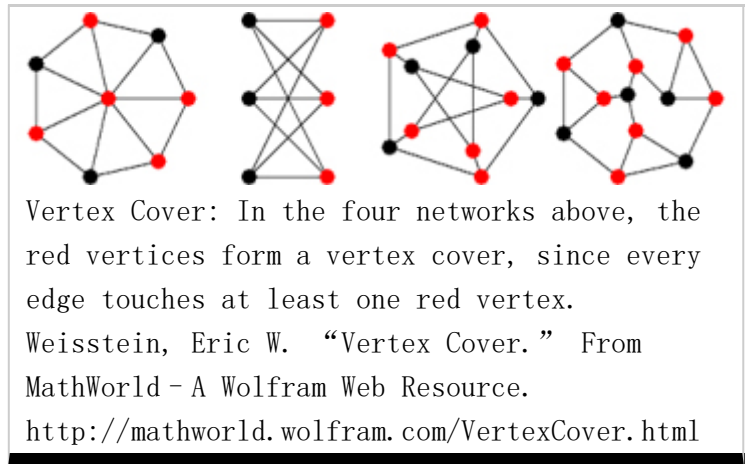
And also like many great conjectures, the UGC has the potential to advance the field of computational complexity regardless of whether it is eventually proved or disproved. If it is proved, it will give exact hardness results for a wide range of approximation problems, and in the case of a wide class of problems known as constraint satisfaction problems, it will show that the ‘obvious suspect’ method for solving these problems truly is the best one, bringing a satisfying closure to that research area. Disproving the UGC might be even more exciting, since it would require the creation of a powerful new approximation algorithm that would presumably be very different from the approximation algorithms known today.

“I do believe whether it is eventually shown to be false or not changes little about the brilliance of the conjecture,” writes Richard Lipton, of Georgia Institute of Technology in Atlanta, in a blog post on the UGC. “It takes insight, creativity of the highest order, and a bit of ‘guts’ to make such a conjecture.”

Exact Hardness

To get a sense of the types of approximation problems on which the UGC has the potential to shed light, let's turn our microscope on a classic problem known as Minimum Vertex Cover, which has applications in communications, civil and electrical engineering, and bioinformatics. Given a network—a collection of nodes (also called vertices) and edges connecting some of the vertices—a vertex cover is a subset of the vertices (called the cover) such that every edge in the network touches at least one vertex in the cover. The Minimum Vertex Cover problem asks for a vertex cover with the smallest possible number of vertices.

One way to find the smallest possible vertex cover is simply to look at *all* subsets of the vertices, weed out the ones that aren't vertex covers, and then see which of the remaining subsets are the smallest. However, for a network with n vertices, there are 2^n subsets to check. Exponential functions such as 2^n grow so quickly that even for fairly small values of n , this algorithm would take longer than the age of the universe for even the fastest computer to run.



The consensus among computer scientists is that for a computer algorithm to be efficient, its running time must be at most a polynomial function of the size of the input, n —that is, something like $2n^2+1$ or $4n^3+n$ (this is, in fact, what it means for an algorithm to belong to the class P). In 1972, Richard Karp of the University of California, Berkeley famously showed that a polynomial-time algorithm for Minimum Vertex Cover does not exist unless $P = NP$: in other words, this problem is NP-hard.

Thus, short of a proof that $P = NP$, the best that researchers can hope for is an efficient *approximation* algorithm: one that is guaranteed to produce vertex covers that are pretty small, but maybe not the smallest.

Luckily, one very simple such algorithm exists: Start by choosing any one edge of your network, and put both its endpoints in your subset. Delete from the network those two vertices and all edges that touch either one of them, and then repeat the process by choosing any one edge from the new, smaller network and putting its two endpoints in your subset. Keep repeating this process until all the edges have been deleted. This will take at most $n/2$ steps, so it is very efficient, unlike the previous algorithm. The resulting subset is a vertex cover, and although it may not be the smallest, it has at most twice as many vertices as it needs, since every vertex cover must have at least one of the two vertices chosen in each round. Computer scientists say this algorithm achieves a factor-of-2 approximation.

Since this algorithm is so simple, it might seem likely that a more ingenious algorithm could do better, coming closer to the smallest number of vertices needed. Indeed, in the

first couple of decades after the P versus NP conjecture was formulated, many computer scientists believed that getting good approximations must be easier than finding the exact answer to an NP-hard question, says Sanjeev Arora, of Princeton University. “The intuition was that there are so many different things you could throw at a problem to get approximate solutions,” he says.

But no one has ever come up with an algorithm that approximates Minimum Vertex Cover to within any constant factor better than 2. Experiences such as this one made computer scientists start to wonder whether there are some problems for which finding a good approximation is every bit as hard as finding an exact solution—that is, NP-hard.

In 1992, computer scientists achieved a breakthrough: the famous PCP theorem, a powerful but very technical result that allowed researchers to prove that some problems are NP-hard even to approximate beyond some factor. Using the PCP theorem, for example, Irit Dinur, currently at the Weizmann Institute of Science in Rehovot, Israel, and Samuel Safra of Tel Aviv University showed in 2002 that assuming P is not equal to NP, there is no efficient algorithm for Minimum Vertex Cover with an approximation factor better than $10^{\sqrt{5}-21}$, or about 1.36—that is, an algorithm guaranteed always to produce a vertex cover at most 36% larger than the minimum size.


Results like this one—in which there is a gap between the approximation factor of the best known algorithm and the approximation factor of the hardness result—create a tantalizing question for researchers. In the case of Minimum Vertex Cover, is the factor-of-2 approximation achieved by the simple algorithm we described indeed the best possible, and the PCP theorem is simply not powerful enough to prove that? Or is there a better algorithm out there, one perhaps that achieves a factor of 1.36?

Such questions are of interest not just to theoretical computer scientists. Improving a chip design algorithm from a factor-of-3 approximation to a factor-of-2 approximation might mean millions of dollars saved in improved efficiency, but it would be useful to know ahead of time if there is no point in searching for such an improvement because none exists.

Thus, once the PCP theorem allowed researchers to prove approximation hardness results, it was natural for computer scientists to “get greedy,” Wigderson says. Computer scientists started to want exact hardness results, that is, theorems of the form ‘It is possible to approximate problem X up to a factor of Y, but doing any better is NP-hard.’

In the late 1990s, in a landmark paper, Johan Håstad, of the Royal Institute of Technology in Stockholm, figured out how to get exact hardness results for a few specific problems. Some researchers managed to adapt his arguments to handle more problems, but for many problems, there were seemingly insurmountable technical obstacles to applying this paradigm.

Sidebar: Efficient Solutions



The P versus NP conjecture concerns the difference between problems that can be solved efficiently and problems, such as trunk packing, that may be hard to solve, but for which it's easy to check efficiently whether a proposed solution is correct.

[read more >](#)

“There were a lot of important problems we just couldn’t do,” says Wigderson.

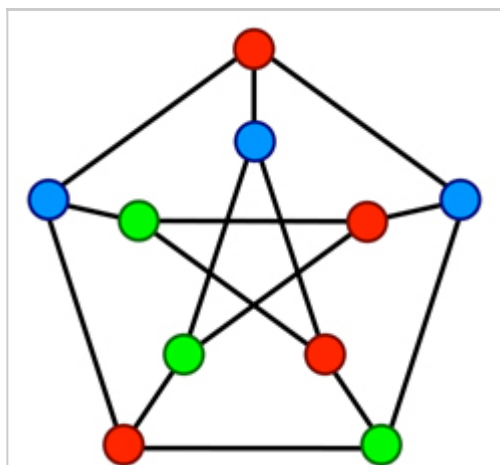
And that’s where matters stood in 2001, when a young Princeton graduate student knocked on his advisor’s door to share a new approximation-hardness idea he had been playing around with.

The Unique Games Conjecture

The student, Subhash Khot, had noticed that a lot of the hard parts about an approximation problem he was working on seemed to be encapsulated in a conjecture about the hardness of assigning colors to the vertices of a network. A few days later, he told his advisor, Sanjeev Arora, that the conjecture was helpful in another problem as well. Arora thought the conjecture was worth writing up, but probably not important enough to submit to the best computer science conferences.

“We had no idea it would become so central,” Arora recalls.

Khot’s Unique Games Conjecture has its roots in the Network Coloring problem: Given a network and a set of colors, is it possible to color the vertices of the network in such a way that two vertices that share an edge always have different colors?



Vertex Coloring: Given a network, is there a way to color the vertices red, green and blue such that vertices that share an edge are always different colors? Figuring out which networks have this property is an NP-hard problem.

In the case of only two colors (say, red and green), it’s easy to answer this question efficiently for any given network: Simply start with a random vertex and color it (say) red. Now all the vertices connected to that one have to be colored green. Color those, and continue to color every vertex connected to an already colored vertex, until you’ve either reached a contradiction or successfully colored the whole network.

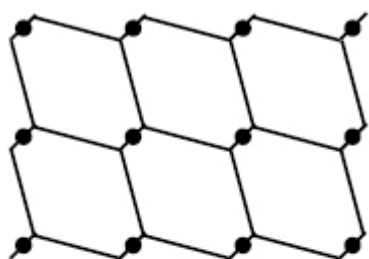
If you are working with more than two colors, choosing the color of one vertex does not force the color of any of the vertices connected to it, so it’s impossible to carry out an algorithm as simple as the one described above. In fact, for more than two colors, deciding which networks can be colored is an NP-hard problem.

Khot decided to bridge the gulf between the two-color algorithm and the situation with more than two colors by considering networks that come equipped with a set of coloring rules, or constraints, such that if two vertices are connected by an edge, then whenever one edge gets colored, the constraint specifies what the color of the other edge must be. For such networks, an algorithm similar to the one for two colors determines, easily enough, whether the network can be colored in a way that satisfies the constraints.

For theoretical computer scientists, the natural next question is the optimization version of this problem: For a network that can’t be colored in a way that satisfies *all* the constraints, which coloring satisfies the *most* constraints possible? We will call this

the Unique Games problem (the name has its origin in a different form of the problem, which is more complicated to state).

Sidebar: Spherical Cubes



Sometimes, even a failed attempt to prove the UGC can bear fruit. When O'Donnell, Kindler and Uriel Feige of the Weizmann Institute tried to get at the UGC using a technique called parallel repetition, they found that the statement they wanted to prove reduced to a simple, elegant question about the structure of foams.

[read more >](#)

Khot conjectured that this innocent-looking question is in some sense the ultimate in hard-to-approximate problems.

More precisely, Khot's Unique Games Conjecture makes the following rather technical statement: For every value of $\epsilon > 0$, no matter how small, there is a number of colors k (possibly very large) for which, looking at constraint networks with k colors, it is NP-hard to tell the difference between networks for which at least $(100 - \epsilon)\%$ of the constraints can be satisfied and networks for which at most $\epsilon\%$ of the constraints can be satisfied.

So for example, choosing $\epsilon = 1$, there is some number of colors k for which it is NP-hard (that is, effectively impossible) to distinguish between networks in which it is possible to satisfy at least

99% of the constraints and networks in which it is possible to satisfy at most 1% of the constraints.

On the face of it, this seems like an improbable assertion. Intuition suggests that constraint networks in which almost all the coloring constraints can be satisfied should look fundamentally different from constraint networks in which almost none of the constraints can be satisfied, and it should be easy to spot the differences between them. Yet when the colors and size of the networks start to number in the billions or trillions, this intuition suddenly starts feeling a lot less obvious.

It follows from the conjecture that the Unique Games coloring problem cannot be approximated efficiently up to *any* constant factor (in contrast with, for example, Minimum Vertex Cover, which you may recall can easily be approximated to within a factor of 2). To see why this is so, we'll show that the Unique Games problem can't be approximated up to a factor of $\frac{1}{2}$. A similar argument (with different numbers) works for any other factor you choose.

So suppose, for a moment, that we *could* approximate Unique Games up to a factor of $\frac{1}{2}$ —that is, we could produce an efficient algorithm that, for any number of colors k and any constraint network, would produce a coloring of that network that satisfied at least half as many constraints as the best possible coloring.

This algorithm would immediately give us an efficient way to distinguish between (for example) networks in which one can satisfy at least 99% of the constraints and networks in which one can satisfy at most 1%: For the former networks, the algorithm would produce a coloring that satisfied at least $99/2$ or 44.5% of the constraints, whereas for the latter networks, by their very nature the algorithm could produce no coloring that satisfied more than 1%. Thus, the existence of such an algorithm would violate the UGC. In other words,

assuming the UGC is true, approximating the Unique Games problem up to a factor of $\frac{1}{2}$ —or indeed any constant factor—is NP-hard.

This reasoning is called a reduction argument: we reduced one problem (distinguishing between the 99% and 1% cases) to another problem (approximating the Unique Games problem up to a factor of $\frac{1}{2}$), and then we used the hardness of the former problem to argue about the hardness of the latter problem. Reductions are the standard approach to proving hardness results in computational complexity.

Perhaps because of the relative simplicity of the coloring problem Khot studied, in the years following his statement of the UGC, computer scientists started noticing that the problem seemed to lend itself to reduction arguments—much more so than the complicated PCP theorem. Thus, it gave researchers a way to make hardness arguments for problems that had previously been beyond their reach.

“The UGC seems to encapsulate and simplify some very hard technical stuff,” O’Donnell says.

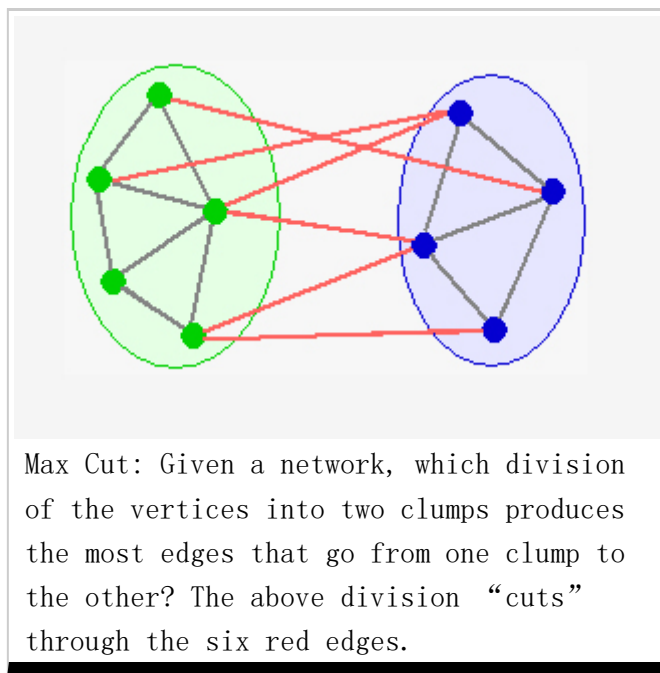
Universal Hardness

The first glimmer of the importance of the UGC came in 2003, when Khot and Oded Regev, of Tel Aviv University and the École Normale Supérieure in Paris, proved that if UGC is true, then computer scientists’ intuition about the Minimum Vertex Cover problem is correct: the problem is NP-hard to approximate up to any constant better than 2.

Two years later, the UGC generated a truly startling finding, one that made computer scientists sit up and take note. It concerned a problem called Max Cut, which asks, given a network, how to divide up the vertices into two subsets in such a way that this division cuts through as many edges as possible (that is, separates the two endpoints of the edge into opposite subsets).

In 1994, Michel Goemans of MIT and David Williamson of Cornell University used a geometric argument involving trigonometric identities to come up with an efficient approximation algorithm for Max Cut that comes within about 87% of the best possible cut. Most computer scientists, says Luca Trevisan of Stanford University, believed that this particular number was an artifact of the geometric arguments involved and that there was probably a better approximation algorithm out there, since Max Cut is not inherently a deeply geometric problem. Hastad had used the PCP theorem to show in 1999 that it is NP-hard to approximate Max Cut better than about 94%, but that left plenty of room to improve on the 87% algorithm.

In 2005, however, Khot, O’Donnell, Guy Kindler of the Hebrew University of Jerusalem and Elchanan Mossel of the Weizmann Institute showed that if UGC is true, then the 87%



algorithm is the best possible efficient approximation to Max Cut.

“This was the defining moment when the Unique Games Conjecture became a very big deal,” Trevisan says. “The conjecture was giving us the exact answer—telling us that this really strange constant that comes from a geometric analysis is the best approximation for this very combinatorial problem.”

Since that time, researchers have proved numerous results of the form ‘If the UGC is true, then problem X is NP-hard to approximate better than a factor of Y.’ And in 2008, while at the University of Washington, Prasad Raghavendra—now of the Georgia Institute of Technology—astonished computer scientists by proving that UGC gives exact hardness results for all constraint satisfaction problems—problems such as the Unique Games problem in which the goal is to fulfill as many of a given set of constraints as possible. “Almost all problems in all disciplines have versions that look like that,” Wigderson says.

What’s more, Raghavendra showed that the UGC implies that the best approximation for any such problem is achieved by a very simple algorithm that uses a paradigm known as semidefinite programming.

“It’s amazing,” Wigderson says. “We understand immediately the hardness of an infinite class of problems by relying only on the one problem Khot postulated is hard. I don’t think anyone—not even he—suspected that it would be so universal.”

Unconditionally True

In a certain sense, the results we’ve just described are like a house of cards: If someone disproves the UGC, they will all collapse (although some weaker versions might survive). As computer scientists started making this observation to Khot, he became determined, O’Donnell says, to show that it is possible to use the ideas in the UGC to prove some unconditional theorems: statements whose conclusions do not depend on the UGC ultimately getting proved.



Taxicab Geometry: The effective distance between two buildings is different for a bird than for a taxicab confined to city streets. The distortion between different ways of measuring distance turns out to be intimately connected to the Unique Games Conjecture.

Khot zeroed in on something called the Goemans-Linial Conjecture, an assertion about the geometry of metric spaces: spaces that come equipped with a metric, that is, a definition of the distance between any two points. While we’re all familiar with everyday, ‘as the crow flies’ distance (which mathematicians call the L_2 metric), many other metrics exist. For example, there’s the taxicab metric, also known as the L_1 metric: it specifies that the distance between two points is the length of the shortest route that follows a combination of North-South and East-West streets. And for some subsets of an L_2 space, the square of the L_2 metric is another metric, called the L_2^2 metric.

A fundamental geometric question is how much distortion occurs if you take objects in a metric space and view them through the lens of a different metric. For example, if you take two

buildings in Manhattan, how much distance would you save if you could fly instead of drive between them?

The Goemans–Linial Conjecture asserts, roughly, that the L_2 metric is a not-too-distorted version of the L_1 metric. Specifically, it says that there is some universal distortion constant K such that if you take any finite collection of points in an L_2 metric space, there's a way to embed them (or in unscientific terms, to plunk them down) in an L_1 space so that the distance between any pair of points has been distorted by at most a factor of K .

Computer scientists have known for years that this conjecture is related to the problem of finding a good approximation algorithm for a problem called Sparsest Cut. This problem asks how to divide the vertices of a network into two clusters of roughly equal size, in such a way that as few edges as possible go from one cluster to the other. Algorithms that divide networks into natural clusters are useful for problems in fields such as computer vision and data analysis, since they give a way to find collections of related objects in large data sets.

The Sparsest Cut problem is NP-hard to solve exactly, so the best computer scientists can hope for is a good approximation. One approach to creating such an algorithm is to try to embed the vertices of the network in an L_1 space in such a way that the vertices are far apart on average, but those vertices connected by an edge are fairly close together. Doing so would give a geometric way to ‘see’ the natural clusters in the network, and computer scientists have shown that this process would lead to a good approximation algorithm for Sparsest Cut. Unfortunately, no one has found an efficient way to create such an embedding.

What computer scientists can do quite well, however, is embed the vertices in an L_2 space. If the Goemans–Linial conjecture could be proven, then computer scientists would know that this embedding was just a distorted version of a similar embedding in an L_1 space. Using this knowledge, they could prove that there was an algorithm that approximated Sparsest Cut up to a constant factor.

In the early years of the new millennium, computer scientists had proved various results that seemed to be leading towards a potential proof of the Goemans–Linial conjecture, and by 2005, excitement was running high. However, working with Nisheeth Vishnoi, currently at Microsoft Research India in Bangalore, Khot found that if UGC is true then Sparsest Cut cannot be approximated up to *any* constant factor. This would imply that the Goemans–Linial conjecture is in fact false.

Since the Goemans–Linial conjecture is a purely geometric statement, Khot and Vishnoi felt that it should be possible to disprove it without depending on the UGC, a statement that is, after all, about computational complexity. In 2005, they managed to do just that, using ideas from the UGC but not relying on the statement of the UGC itself.

“They proved that this approach to Sparsest Cut won’t work,” O’Donnell says. “It was a bit of a blow, since it was the only method we knew, and people had high hopes for it.”

This finding is not the only unconditional result to emerge from studying the UGC. The conjecture has led researchers to prove fundamental results about voting systems and the

structure of foams (see, respectively, the sidebars *Stable Elections* and *Spherical Cubes*).

“Some very natural, intrinsically interesting statements about things like voting and foams just popped out of studying the UGC,” O’Donnell says. “Even if the UGC turns out to be false, it has inspired a lot of interesting math research.”

True or false?

While the unconditional results that have emerged from the UGC have already proved its value to the mathematics and computer science community, it would nevertheless be nice (to put it mildly) to know whether it is in fact true.

For the past five years, computational complexity researchers have been working away at this question, but without a huge amount of progress, O’Donnell says.

“Half of the people are working on proving it and half on disproving it, and they’re both failing,” he says. “That points to the UGC’s status as a very hard problem.”

What has emerged is a jumble of intriguing, halfway-there results. One finding, for example, says roughly that if the UGC is indeed true, it takes a very, very large number of colors to make a Unique Games network coloring hard to approximate, much more so than in corresponding problems involving the PCP theorem. Researchers have also shown that, unlike with many problems, randomly chosen networks are not hard to approximate and so will not be an avenue to proving the UGC.

And last year, Arora, together with Boaz Barak and David Steurer of Microsoft Research New England, proved that there is a sub-exponential algorithm for Unique Games—that is, one that is faster than an exponential algorithm, but still not as fast as a polynomial algorithm. That means that Unique Games joins a small and exclusive clique of problems for which the best known algorithm falls between exponential and polynomial time. “That was a surprise,” Arora says.

The assortment of findings about the UGC are reminiscent of the allegory in which a collection of blind people touch different parts of an elephant and each get a different impression of the elephant’s nature, says Dana Moshkovitz of MIT.

“We know all these weird things about the UGC,” she says. “But maybe there’s a proof that would explain away all of the weird things.”

If someone does unveil the UGC’s inner elephant and comes up with a proof some day, it will be a cause for celebration, since that would mean (among many other things) that

Sidebar: Stable Elections



In a typical election, there’s a reasonable chance that some votes will be misrecorded. How likely are misrecorded votes to alter the outcome, and how does that likelihood depend on which voting system is being used?

[read more >](#)

constraint-satisfaction problems have been thoroughly understood. “It would very much complete the theory,” O’Donnell says.

At the same time, if someone can disprove the UGC, it might give researchers even more to think about. “People suspect that to show the UGC is wrong, someone would have to come up with an amazing algorithm involving a lot of new ideas,” Arora says.

Unlike some other famous conjectures such as the Riemann Hypothesis and the P versus NP problem, for which most mathematicians agree about the expected answer, the UGC’s believers and disbelievers form two pretty even camps, O’Donnell says. “This may be an open problem for a very long time,” he says.



Sidebar: Efficient Solutions

As anyone who has gone on a cross-country road trip will know, trying to fit a large assortment of objects into the trunk of a car may require many attempts, a very long time, and a bit of divine inspiration. By contrast, if someone else is kind enough to pack your trunk for you, checking whether he has accidentally left any items on the curb takes a matter of seconds.

The P versus NP conjecture concerns the difference between problems that can be solved efficiently and problems, such as trunk packing, that may be hard to solve, but for which it’s easy to check efficiently whether a proposed solution is correct.

Efficiency is an intuitively clear concept, but what precisely does it mean for a computational problem to have an efficient solution? One thing it does *not* mean is that every instance of the problem can be solved in a reasonable amount of time. For most computational problems, the length of time it takes to get an answer increases as the size of the ‘input’ — the lengths of some numbers we are trying to add, or the number of objects we are trying to fit in a suitcase—grows. For even a simple problem like adding two numbers, there are numbers long enough that adding them together would take longer than a year, or a million years, or any length of time we’d care to specify.

In practice, though, we never need to add numbers that large. To be practical, an algorithm does not have to solve every instance of a problem quickly—just those instances with reasonable input sizes (something that will vary from problem to problem). As long as the running time of the algorithm doesn’t grow too fast as the size of the input grows, the algorithm should be able to handle the practical examples we would actually want to throw at it. But how fast is too fast?

In the 1960s, computer scientists decided that the most reasonable definition of not-too-fast growth is what is called polynomial time. An algorithm is said to run in polynomial time if, for some constants C and k , the algorithm takes at most Cn^k steps to solve any instance of the problem with inputs of size n .

It’s not hard to show that this concept is independent of which computational model we use: if a problem has a polynomial-time algorithm on one type of computer, that algorithm

can be translated into a polynomial-time algorithm on any other type of computer. Computer scientists have named the class of problems that can be solved by polynomial-time algorithms class P.

Polynomial functions grow comparatively slowly—much, much more slowly than exponential functions such as 2^n , which typically appear, for example, as the running time of brute-force search algorithms. Algorithms whose running times grow like exponential functions are so slow that they are impractical for all but the very smallest inputs. And while an algorithm whose running time is bounded by a large polynomial such as n^{100} may not be very practical to run either, such algorithms don't seem to happen a whole lot, for some reason: for most of the problems known to be in P, computer scientists have found algorithms whose running time is bounded by a polynomial with a small exponent, such as n^3 or n^4 . Thus, most computer scientists take the view that for a problem to belong to class P is equivalent to it being feasible to solve.

The problem of adding two numbers belongs to P, since the standard algorithm schoolchildren learn is efficient (though it may not seem so when being carried out by a seven-year-old): Its running time is proportional to the length of the two numbers. By contrast, computer scientists have not so far been able to find a polynomial-time algorithm to solve the trunk packing problem. Thus, this problem may not belong to class P; however, as we have seen, it does belong to another class, which computer scientists call NP: the class of problems for which there is a polynomial-time algorithm to verify whether a proposed solution is indeed correct.

Any problem in P is automatically in NP, but what about the reverse? In the early 1970s, Stephen Cook, currently at the University of Toronto, and Leonid Levin, currently at Boston University, independently formulated the famous P versus NP conjecture: that these two classes are not the same. In other words, there are problems in NP that can never be solved in polynomial time by any computer algorithm, no matter how clever.

In a sense, the conjecture asserts that creativity cannot be automated—that there is a fundamental gulf between the ability to solve a problem and the ability merely to appreciate someone else's solution. If the conjecture is false, a proof of that fact would have profound implications for a wide range of disciplines.

For example, in mathematics, it is a simple matter to create an efficient algorithm that checks the correctness of any proposed mathematical proof. A proof that $P = NP$ would allow mathematicians to transform this algorithm into an efficient algorithm to *come up with* a proof of any true mathematical statement. If this could be done, the field of mathematics would be transformed from its current form into something completely unrecognizable.

To many mathematicians, it seems unfathomable that mathematical insight could be automated in this way, and indeed the majority of mathematicians and theoretical computer scientists believe that P does not equal NP . Yet the conjecture has held out against all attempts to prove or disprove it over the last 40 years.

Sidebar: Spherical Cubes

Sometimes, even a failed attempt to prove the UGC can bear fruit. When O’ Donnell, Kindler and Uriel Feige of the Weizmann Institute tried to get at the UGC using a technique called parallel repetition, they found that the statement they wanted to prove reduced to a simple, elegant question about the structure of foams.

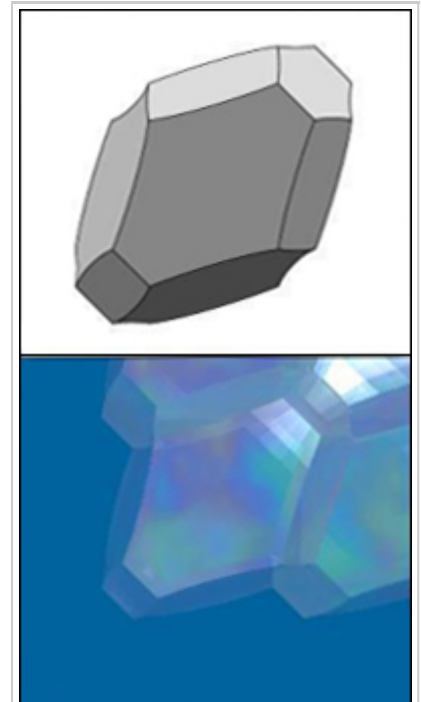
The most basic question about foams, which has been studied for more than 200 years, asks, for each dimension d , for the shape with the smallest surface area that divides up d -dimensional space into bubbles of volume 1. This problem was solved for dimension 2 in 1999 by Thomas Hales of the University of Pittsburgh—the best shape is simply the regular hexagon, although proving that fact was no easy matter. In dimension 3, a candidate shape has been identified, but a proof has eluded researchers. In higher dimensions, even less is known.

The researchers’ attempt to prove the UGC boiled down to a question closely related to the basic foam problem, which we’ll call the integer lattice foam problem: If the allowable bubble shapes are limited to those that can tile space by shifting copies of the bubble whole numbers of steps along the coordinate axes, which shape has the lowest possible surface area?

So for example, in dimension 2, a square would be a viable candidate, since it tiles the plane by shifts to the left and right, up and down. The regular hexagon, however, would not be a candidate, since to fill space with regular hexagons involves some diagonal shifts of the hexagon, and some horizontal shifts that are not by whole numbers. It turns out, though, that the best integer lattice tiling is in fact a hexagon, though not a regular one: In 1989, Jaigyoung Choe of the Korea Institute of Advanced Study in Seoul proved that the best shape is an irregular hexagon whose perimeter is about 3.86, slightly edging out the square, whose perimeter is 4.

A d -dimensional cube has $2d$ sides, so the cube of volume 1 has surface area equal to $2d$. Thus, the shape that solves the integer lattice foam problem in dimension d must have surface area at most $2d$ (since otherwise the cube would beat it out). And since the shape of volume 1 with the lowest surface area overall is the sphere, the solution to the problem must have surface area at least that of the sphere, which in large dimensions is roughly proportional to the square root of d .

The researchers expected—and needed for their attempted proof of the UGC—for the solution to the integer lattice problem to have surface area roughly proportional to that of a cube (that is, $2d$) when d is a large number. To their surprise, however, O’ Donnell and



Spherical Cube: While studying the UGC, Kindler, O’ Donnell, Rao and Wigderson discovered this three-dimensional shape, which tiles all of space when copies of the shape are placed along a cubical lattice. Of all known shapes with this property, the shape above has the lowest surface area..

Kindler—together with Wigderson and Anup Rao, currently at the University of Washington in Seattle—found that for large dimensions d , there exist shapes that tile space by whole-number steps, and whose surface area is roughly proportional to the square root of d —in other words, their surface area behaves more like that of a sphere than that of a cube.

“There are shapes that are almost spherical but still tile space in a cubical pattern, which is rather counterintuitive,” O’Donnell says. The fact that the UGC led researchers to such an intrinsically interesting, natural question about foams “adds to its mystique,” he says.

Sidebar: Stable Elections

In a typical election, there’s a reasonable chance that some votes will be misrecorded. How likely are misrecorded votes to alter the outcome, and how does that likelihood depend on which voting system is being used? This question is not just theoretical: In the 2000 Presidential Election, after all, the outcome hinged on whether a handful of votes in Florida had been counted correctly. Had plurality voting been the law of the land, instead of the Electoral College system, the outcome would not have been in dispute: Everyone agrees that in the national vote, Al Gore won by more than half a million votes.

The voting methods that are the most stable with respect to misrecorded votes—that is, the least likely to flip outcomes if a few votes are misrecorded—are rather unsavory ones. A system that always chooses the candidate selected by Voter A, or that always awards the win to Candidate 1, will seldom or never be affected by a misrecorded vote. Since no one presumably would choose such a system for a democratic election, it makes sense to ask which voting system is stablest among those systems that give all voters roughly equal weight and are not biased toward or against any particular candidate.

When there are two candidates, this problem is, in fact, related to the Sparsest Cut problem. Imagine a network in which each vertex stands for one possible combination of candidate choices by all the voters (so if there are n voters, this network will have 2^n vertices). If your election is happening in a setting in which, say, about 1% of the votes are typically misrecorded, then connect two vertices by an edge if they are identical except for about 1% of the voters.

A voting system is simply a rule that assigns a winner to each vertex in the network. Thus, it divides the network into two subsets: the vertices that result in a win for Candidate 1, and the vertices that result in a win for Candidate 2. These two sets will be roughly equal in size, assuming we’ve left out the unfairly slanted voting systems discussed above.

For the voting system to be stable with regard to misrecorded votes, it should assign the same outcome to both the endpoints of each edge—in other words, it should not put vertices that are connected by an edge into opposite subsets. Thus, looking for the most stable voting system corresponds to looking for the sparsest cut of the network.

It was actually in the course of analyzing Max Cut in 2004 that Khot, O’Donnell, Kindler and Mossel were led to examine the stability of voting systems. (While Max Cut and Sparsest

Cut seem like opposite problems, they use a lot of the same ideas and techniques, O’ Donnell notes.)

The researchers found that when they plugged Max Cut into the framework of the UGC, the statement about 87% being the best approximation for Max Cut boiled down to proving a simple statement about voting systems: namely, that if the two types of extremely unbalanced voting systems are excluded from consideration, then majority voting is the stablest system. The following year, O’ Donnell, Mossel and Krzysztof Oleszkiewicz of Warsaw University were able to prove this “Majority is Stablest” theorem, vindicating the intuition from the 2000 Presidential Election.

“Other people had thought about voting stability before, but the UGC really narrowed us in on the exact statement and gave us a strong motivation for proving it,” O’ Donnell says.



Further Reading

1. S. Arora, B. Barak and D. Steurer. Subexponential algorithms for unique games and related problems. In *Proc. 51st IEEE Symposium on Foundations of Computer Science, 2010*.
2. U. Feige, G. Kindler, and R. O’Donnell. Understanding parallel repetition requires understanding foams. In *Proceedings of the Annual IEEE Conference on Computational Complexity*, pages 179-192, 2007.
3. J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, **48**: 798-859, 2001.
4. S. Khot. On the power of unique 2-prover 1-round games. In *Proc. 34th ACM Symposium on Theory of Computing*, 2002.
5. S. Khot. On the Unique Games Conjecture. In *Proc. 25th IEEE Conference on Computational Complexity*, 99-121, 2010.
6. S. Khot, G. Kindler, E. Mossel, and R. O’Donnell. Optimal inapproximability results for MAX-CUT and other two-variable CSPs? In *Proc. 45th IEEE Symposium on Foundations of Computer Science*, 146-154, 2004.
7. S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. In *Proc. 18th IEEE Conference on Computational Complexity*, 2003.
8. S. Khot and N. Vishnoi. The unique games conjecture, integrability gap for cut problems and embeddability of negative type metrics into ℓ_1 . In *Proc. 46th IEEE Symposium on Foundations of Computer Science*, 2005.
9. G. Kindler, R. O’Donnell, A. Rao, and A. Wigderson. Spherical cubes and rounding in high dimensions. In *Proc. Annual IEEE Symposium on Foundations of Computer Science*, 189-198, 2008.
10. R. Lipton, Unique Games: A Three Act Play. In blog *Gödel’s Lost Letter and P=NP*, May 5, 2010; <http://rjlipton.wordpress.com/2010/05/05/unique-games-a-three-act-play/>
11. E. Mossel, R. O’Donnell, and K. Oleszkiewicz. Noise stability of functions with low influences: invariance and optimality. *Annals of Mathematics* **171** no.1, 295-341, 2010.
12. P. Raghavendra. Optimal algorithms and approximability results for every CSP? In *Proc. ACM Symposium on the Theory of Computing*, 245-254, 2008.
13. L. Trevisan. On Khot’s Unique Games Conjecture. *AMS Current Events Bulletin*, 1-21, 2011.

