Structure vs Combinatorics in Computational Complexity

Boaz Barak

October 7, 2013

Computational problems come in all different types and from all kinds of applications, arising from engineering as well the mathematical, natural, and social sciences, and involving abstractions such as graphs, strings, numbers, and more. The universe of potential algorithms is just as rich, and so a priori one would expect that the best algorithms for different problems would have all kinds of flavors and running times. However natural computational problems "observed in the wild" often display a curious *dichotomy*— either the running time of the fastest algorithm for the problem is some small polynomial in the input length (e.g., $O(n)$ or $O(n^2)$) or it is *exponential* (i.e., $2^{\epsilon n}$ for some constant $\epsilon > 0$). Moreover, while indeed there is a great variety of efficient algorithms for those problems that admit them, there are some general principles such as *convexity* (i.e., the ability to make local improvements to suboptimal solutions or local extensions to partial ones) that seem to underly a large number of these algorithms.[1] This phenomenon is also related to the "unreasonable effectiveness" of the notion of NP-completeness in classifying the complexity of thousands of problems arising from dozens of fields. While a priori you would expect problems in the class NP (i.e., those whose solution can be efficiently certified) to have all types of complexities, for natural problems it is often the case that they are either in P (i.e., efficiently solveable) or are NP-hard (i.e., as hard as any other problem in NP, which often means complexity of $2^{\epsilon n}$, or at least 2^{n^ϵ}).

To be sure, none of these observations are universal laws. In fact there are theorems showing exceptions to such dichotomies: the Time Hierarchy Theorem says that for essentially any time-complexity function $T(\cdot)$ there is a problem whose fastest algorithm runs in time (essentially) $T(n)$. Also, Ladner's Theorem says that, assuming P$\neq$NP, there are problems that are neither in P nor are NP-complete. Moreover, there are some *natural* problems with apparent "intermediate complexity". Perhaps the most well known example is the *Integer Factoring* problem mentioned below.

[1] The standard definition of "convexity" of the solution space of some problem only applies to continuous problems and means that any weighted average of two solutions is also a solution. However, I use "convexity" here in a broad sense meaning having some non-trivial ways to combine several (full or partial) solutions to create another solution; for example having a matroid structure, or what's known as "polymorphisms" in the constraint-satisfaction literature.

Nevertheless, the phenomenon of dichotomy, and the related phenomenon of recurring algorithmic principles across many problems, seem far too prevalent to be just an accident, and it is these phenomena that are the topic of this essay.

I believe that one reason underlying this pattern is that many computational problems, in particular those arising from combinatorial optimization, are *unstructured*. The lack of structure means that there is not much for an algorithm to exploit and so the problem is either "very easy"— e.g., the solution space is simple enough so that the problem can be solved by local search or convex optimization[2]— or it is "very hard"— e.g., it is NP-hard and one can't do much better than exhaustive search. On the other hand there are some problems that posses a certain (often algebraic) *structure*, which typically is exploitable in some non-trivial algorithmic way. These structured problems are hence never "extremely hard", but they are also typically not "extremely easy" since the algorithms solving them tend to be more specialized, taking advantage of their unique properties. In particular, it is harder to understand the complexity of these algebraic problems, and they are more likely to yield algorithmic surprises.

I do not know of a good way to formally classify computational tasks into *combinatorial/unstructured* vs. *algebraic/structured* ones, but in the rest of this essay I try to use some examples to get a better sense of the two sides of this divide. The observations below are not novel, though I am not aware of explicit expositions of such a classification (and would appreciate any pointers, as well as any other questions or critique). As argued below, more study into these questions would be of significant interest, in particular for cryptography and average-case complexity.

Combinatorial/Unstructured problems

The canonical example of an unstructured combinatorial problem is SAT — the task of determining, given a Boolean formula φ in variables $x_1, \ldots, x_n$ with the operators $\neg, \wedge, \vee$, whether there exists an assignment x to the variables that makes $\varphi(x)$ true. SAT is an NP-complete problem, which means it cannot be solved efficiently unless P=NP. In fact, the Exponential Time Hypothesis posits that every algorithm solving SAT must take at least $2^{\epsilon n}$ time for some $\epsilon > 0$. SAT illustrates the above dichotomy in the sense that its natural restrictions are either as hard as the general, or become easily easily solvable, as in the case of the 2SAT problem (where the formula is in conjunctive normal form with each clause of arity 2) that can be solved efficiently via a simple propagation algorithm. This observation applies much more generally than SAT. In particular the widely believed Feder-Vardi dichotomy conjecture states that

[2]Of course even if the algorithm is simple, analyzing it can be quite challenging, and actually obtaining the fastest algorithm, as opposed to simply one that runs in polynomial time, often requires additional highly non-trivial ideas.

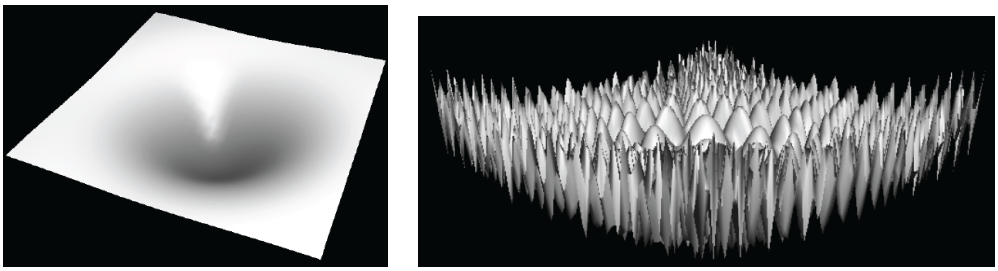

Figure 1: An illustration of the solution space geometry of a random SAT formula, where each point corresponds to an assignment with height being the number of constraints violated by the assignment. The left figure depicts the "ball" regime, where a satisfying assignment can be found at the bottom of a smooth "valley" and hence local algorithms will quickly converge to it. The right figure depicts the "shattered" regime where the surface is very ragged, with an exponential number of crevices and local optima, thus local algorithms (and as far as we know any algorithm) will likely fail to find a satisfying assignment. Figures courtesy of Amin Coja-Oghlan.

every constraint satisfaction problem (CSP) is either NP hard or in P. In fact, researchers conjecture (and have partially confirmed) the stronger statement that every CSP can either be solved by some specific algorithms of low polynomial-time (either propagation or generalizations of Gaussian elimination) or is NP hard via a linear blowup reduction from SAT, and hence (under the Exponential Time Hypothesis) cannot be solved faster than $2^{\epsilon n}$ time for some $\epsilon > 0$.[3]

Random SAT formulas also display a similar type of dichotomy. Recent research into random k-SAT (based also on tools from statistical physics) suggests that they have multiple *thresholds* where the problem changes its nature. When the *density* α (i.e., ratio of constraints to variables) of the formula is larger than some number α_s (equal roughly to $2^k \ln 2$) then with high probability the formula is "overconstrained" and no satisfying assignment exists. There is some number $\alpha_d < \alpha_s$ (equal roughly to $2^k \ln k/k$), such that for $\alpha < \alpha_d$, the space of satisfying assignments for a random formula looks roughly like a discrete ball, and, due to this relatively simple geometry some local-search type algorithms can succeed in finding satisfying assignments. However for $\alpha \in (\alpha_d, \alpha_s)$, satisfying assignments still exist, but the geometry of the solution space becomes vastly different, as it *shatters* into exponentially many clusters, each such cluster separated from the others by a sea of assignments that violate a large number of the constraints, see Figure 1. In this regime no efficient algorithm

[3]The main stumbling block for completing the proof is dealing with those CSPs that require a Gaussian-elimination type algorithm to solve; one can make the argument that those CSP's actually belong to the algebraic side of our classification, further demonstrating that obtaining precise definitions of these notions is still a work in progress. Depending on how it will be resolved, the Unique Games Conjecture, which I discussed here, might also give rise to CSP's with "intermediate complexity" in the realm of *approximation algorithms*. Interestingly, both these issues go away when considering random, noisy, CSP's, as in this case solving linear equations becomes hard, and solving Unique Games becomes easy.

is known to find the satisfying assignment, and it is possible that this is inherently hard.[4]

Dichotomy means that when combinatorial problems are hard, then they are typically very hard, not just in the sense of not having a subexponential algorithm, but they also can't be solved non-trivially in some intermediate computational models that are stronger than P but cannot solve all of NP such as quantum computers, statistical zero knowledge, and others. In particular for combinatorial problems (quoting Lovász) the existence of a *good characterization* (i.e., the ability to efficiently *verify* both the existence and non-existence of a solution) goes hand-in-hand with the existence of a good algorithm. Using complexity jargon, in the realm of combinatorial optimization it seems to hold that P=NP∩coNP, even though we believe this is false in general. Indeed, for many combinatorial problems such as matching, max flow, planarity, etc.. demonstrating a good characterization is an important step toward finding an efficient algorithm. This is related to the notion of *duality* in convex programming, which is often the method of choice to solve such problems.

Combinatorial problems can be quite useful for *cryptography*. It is possible to obtain one-way functions from random instances of combinatorial problems such as SAT and Clique. Moreover, the problem of attacking a cryptographic primitive such as a block cipher or a hash function can itself be considered a combinatorial problem (and indeed this connection was used for cryptanalysis). However, these are all *private key* cryptographic schemes, and do not allow two parties to communicate securely without first exchanging a secret key. For the latter task we need public key cryptography, and as we discuss below, the currently known and well-studied public key encryption schemes all rely on *algebraic* computational problems.

Algebraic/Structured problems

Factoring is a great example of an algebraic problem; this is the task of finding, given an n-bit integer N, the prime numbers $p_1, \ldots, p_k$ such that $N = p_1 \cdots p_k$. No polynomial time algorithm is known for Factoring, but it had seen some non-trivial algorithmic advances. While the natural trial-division algorithm takes roughly $2^{n/2}$ steps to solve Factoring, the Number Field Sieve algorithm, which is the current best, takes roughly $2^{n^{1/3} polylog(n)}$ steps. Factoring can also be solved in polynomial-time on *quantum computers* using Shor's Algorithm. Finally, Factoring (or more accurately, the decision problem obtained by looking at individual bits of the output)

[4] The Survey Propagation Algorithm is a very interesting algorithm that arose from statistical physics intuition, and is experimentally better than other algorithms at solving random k-SAT formulas for small k such $k = 3, 4$. However, it is believed, that at least for larger k, it too cannot succeed in the regime where the solution space geometry shatters. The best known algorithm for random k-SAT for large k is given in this paper of Amin Coja-Oghlan.

is also in the class NP∩coNP, which means that one can efficiently *verify* the value of a particular bit of the answer, no matter if this value is zero or one. These results almost certainly mean that FACTORING is *not* NP complete.

There is another, more subjective sense, in which I find FACTORING different from SAT. I personally would be much more shocked by a $2^{\sqrt{n}}$-time algorithm for SAT than by a $2^{n^{1/6}}$-time algorithm for FACTORING. The reason is that, while people have found clever ways to speed up the 2^n time exhaustive search algorithm for SAT (especially on certain types of instances), these approaches all seem to inherently require exponential time, and are not as qualitatively different from exhaustive search in the way that the number field sieve is different from trial division. In contrast, FACTORING clearly has strong algebraic structure that we do not completely understand, and perhaps have not reached the limit of its exploitation by algorithms. To see that this is not completely implausible, consider the problem of computing the discrete logarithm in fields of small characteristic. This problem shares many properties with FACTORING, and it also shared the property of having a best-known running time of $2^{n^{1/3}polylog(n)}$ until this was recently improved to $2^{n^{1/4}polylog(n)}$ and then to $2^{polylog(n)}$.

Not all algebraic problems are hard. Factoring univariate polynomials over finite fields can be solved efficiently using the Berlekamp or Cantor-Zassenhaus algorithms (see e.g. chapter 21 here). This algorithm also exemplifies the statement above, that algorithms for algebraic problems are often very specialized and use non-trivial properties of the problem's structure. For this reason, it's harder to predict with confidence what is the best algorithm for a given algebraic problems, and over the years we have seen several surprising algorithms for such problems, including, for example, the fast matrix multiplication algorithms, the non-trivial factoring algorithms and deterministic primality testing, as well as the new algorithm for discrete logarithm over small-characteristic fields mentioned above.

Relation to cryptography. Algebraic problems are very related to *public key cryptography*. The most widely used public key cryptosystem is RSA, whose security relies on the hardness of FACTORING. The current subexponential algorithms for FACTORING are the reason why we use RSA keys of 1024 or 2048 bits, even though even the yet-to-built exaflop supercomputers would take thousands of years to perform, say, 2^{100} computational operations. This also demonstrates how fragile is RSA to any surprising algorithmic advances. If the exponent of the best factoring algorithm would halve (i.e., change from 1/3 to 1/6) then, roughly speaking, to get equivalent security we would need to *square* the size of the key. Since the RSA encryption and decryption algorithms take time which is at least quadratic in the size of the key, that would make RSA pretty impractical.

Cryptosystems based on the discrete logarithm problem in *elliptic curves* yield one alternative to RSA which currently is not known to be broken in subexponential time. Elliptic-curve discrete log is of course also very much an algebraically structured

problem, and so, I would argue, one in which further algorithmic surprises are hard to rule out. Moreover, like factoring, this problem can be solved in polynomial time by *quantum computers*, using Shor's algorithm.

The only other public key cryptosystems that are researched enough to have some confidence in their security are based on decoding problems for linear codes or integer lattices. These problems are not known to have subexponential algorithms, classical or quantum. Moreover, some variants of these problems are actually *NP-hard*. Specifically, theses problem are parameterized by a number α which is the *approximation factor*, where smaller α means the problem is harder. For example, the shortest vector problem in a lattice can be solved efficiently for $\alpha \geq c^n$ (where $c > 1$ is some constant and n is the dimension of the lattice, which is related to the length of the input), and the problem is NP hard for $\alpha \leq n^\delta$ (where $\delta = \delta(n)$ is some function of n tending slowly to zero). For this reason lattice problems were once seen as a potential approach to getting both private and public crypto based on the minimal assumption that P$\neq$NP, which in particular would yield public key crypto based on unstructured problems such as SAT. However, we only know how to get public key crypto from these problems for $\alpha = n^e$ for some $e > 1/2$ while we have reason to believe that for $\alpha > n^{1/2}$ the problem *does* actually possess algebraic (or at least geometric) structure; this is because in this range the problem has a "good characterization" (i.e., in NP$\cap$coNP or AM$\cap$coAM). A similar phenomenon also occurs for other problems such as learning parity with noise and random 3SAT (see discussion in this paper with Applebaum and Wigderson)— there seem to be two *thresholds* $\alpha_G < \alpha_E$ such that for $\alpha < \alpha_G$ the problem is hard and arguably unstructured, for $\alpha \in (\alpha_G, \alpha_E)$ the problem becomes useful for public key cryptography, but also seems to suddenly obtain some *structure* such as a "good characterization", while for $\alpha > \alpha_E$ the problem becomes easy. Another sign of potential structure in lattice problems is the existence of a subexponential quantum algorithm for the hidden subgroup problem in dihedral groups, which is related to these problems.

The bottom line is that based on the currently well studied schemes, structure is strongly associated with (and perhaps even implied by) public key cryptography. This is troubling news, since it makes public key crypto somewhat of an "endangered species" that could be wiped out by a surprising algorithmic advance. Therefore the question of whether structure is *inherently necessary* for public key crypto is not only of mathematical interest but also of practical importance as well. Cryptography is not just an application of this classification but also provides a useful lens on it. The distinction between private key and public key crypto mirrors the distinction between unstructured and structured problems. In the *private key* world, there are many different constructions of (based on current knowledge) apparently secure cryptosystems; in fact, one may conjecture (as was done by Gowers) that if we just combined a large enough number of random reversible local operations then we would obtain a secure block cipher. In contrast, for *public key* cryptography, finding a construction that

strikes the right balance between structure and hardness is a very hard task, worthy of a Turing award, and we still only know of a handful or so such constructions.

A different approach to average case complexity

I am particularly interested in this classification in the context of *average-case complexity*. In the case of *worst-case* complexity, while we have not yet managed to prove that P≠NP, complexity theorists achieved something like the next best thing—classifying a large number of problems into hard and easy ones based on this single assumption. We have not been able to replicate this success in average case complexity, and there is a good reason for that. Our main tool for basing one assumption on another one— the *reduction*— is extremely problematic in average case complexity, since there are inherent reasons why a reduction would not preserve the distribution of the inputs. To illustrate this, suppose that we tried to show that an average-case problem A is no harder than an average-case problem B using a standard Karp reduction f (i.e., $f : \{0,1\}^n \to \{0,1\}^m$ is a function mapping an A-input x into a B-input y such that $B(y) = A(x)$). For simplicity, assume that the input distribution for both problems is the uniform distribution. This would imply that for a random $x \in \{0,1\}^n$, $f(x)$ should be distributed close to the uniform distribution over $\{0,1\}^m$. But we cannot expect this to happen in any reasonable reduction, as all of them add gadgets or blow up the size of the instance in some way, meaning that $m > n$, in which case $f(x)$ is distributed over a subset of $\{0,1\}^m$ of size less than 2^{m-1} and hence is far from the uniform distribution.[5]

This difficulty is one reason why the theory of average-case complexity is much less developed than the theory for worst-case complexity, even though average-case complexity is much more relevant for many applications. The observations above suggest that at least for combinatorial problems, we might hope for a different approach: define a *meta conjecture* that stipulates that for a whole class of average-case problems, a certain algorithmic framework yields the optimal efficient algorithm, meaning that beating the performance of that algorithm would be infeasible (e.g., take exponential time). To make things more concrete, consider the following hypothesis from a paper

[5] As further argument that reductions should increase the input length, note that if A and B were shown equivalent by reductions f and g that *shrink* the size of the input even by a single bit, then by repeating these reductions recursively shows that both A and B can be solved in polynomial time. This argument can be extended to the case that f and g are length preserving, under the assumption that $f \circ g$ is not too close to the identity permutation and that the inputs of length $n-1$ are embedded in the set of inputs of length n. One can also use similar arguments to rule out certain types of *probabilistic* reductions, even those that increase the input size, if we assume the reduction is efficiently invertible.

with Kindler and Steurer:

> **Random CSP Hypothesis.** For every predicate $P : \{0,1\}^l \to \{0,1\}$, if we let RANDOM MAX($P$) be the problem of estimating the fraction of constraints that can be satisfied for an instance chosen at random, then no efficient algorithm can obtain a better approximation to RANDOM MAX(P) than $\alpha(P)$, where $\alpha(P)$ is the approximation obtained by the canonical semidefinite program (a type of convex relaxation) to this problem.[6]

Note that this is a much more general conjecture that P$\neq$ NP, which can be reduced to the statement that a single problem (say worst-case SAT) cannot be efficiently solved. In contrast, the Random CSP Hypothesis contains an unbounded number of hardness conjectures (one for every predicate) that (except in very special cases) are not known to be reducible to one another. Of course, to derive a concrete assumption about a predicate P from this hypothesis one needs to calculate $\alpha(P)$, but fortunately for random CSP's this can be done easily— one can of course run the algorithm, but there is also an analytical expression for this quantity.

Despite it being such a general hypothesis, I don't think the Random CSP Hypothesis is yet general enough— there may well be significant extensions to this hypothesis that are still true, involving combinatorial problems different than CSP's, and distributions different than the uniform one. Perhaps with time, researchers will find the "right" meta conjecture which will capture a large fraction of the problems we consider "combinatorial".

At first brush, it might seem that I'm suggesting to trivialize research in average-case complexity by simply assuming all the hardness results we wish for. But of course, there is still a very real challenge to find out if these assumptions are actually true! Given our current state of knowledge, I don't foresee an unconditional proof of these types of assumptions, or even a reduction to a single problem, any time soon. But, as I discussed before this doesn't mean we can't gather *evidence* on these meta assumptions. However, such assumptions form very "fat targets" for potential refutations. For example, all we have to do to refute the Random 3CSP Hypothesis is to find a single predicate P and a single efficient algorithm A such that A gives a better approximation factor than $\alpha(P)$ for RANDOM MAX(P). In fact, there are very natural candidate algorithms to do just that, including in particular more complicated convex programs known as semidefinite programming *hierarchies*. Analyzing the performance of such algorithms raises some fascinating mathematical questions, many of which we haven't yet been able to solve, and this is a very interesting research area

[6] The notion of "chosen at random" roughly corresponds to the uniform distribution over inputs, or the uniform distribution with an appropriately "planted" satisfying assignment, with the precise notion of "estimation" being the appropriate one for these different models; see the paper for details. The Random CSP Hypothesis deals with the *overconstrainted* regime of random SAT formulas, as opposed to the *underconstrained* regime in discussed above in the the context of phase transitions.

in its own right. With effort and time, if no refutation is found, we might gain confidence in the veracity of such meta assumptions, and obtain a much clearer view of the landscape of average-case complexity, and complexity at large.

Conclusions

While much of what I discussed consists of anecdotal examples, I believe that some works, such as those related to the Feder-Vardi conjecture or to phase transitions in random CSP's, offer a glimpse of a potential general theory of the complexity of combinatorial problems. I think there is room for some ambitious conjectures to try to illuminate this area. Some of these conjectures might turn out to be false, but we can learn a lot from exploring them. Understanding whether the "markers of structure" such as subexponential algorithms, quantum algorithms, good characterization, usefulness for public key cryptography, etc.. need always go together would be extremely useful for many applications, and in particular cryptography. Even more speculatively, perhaps thinking about these issues can help towards the goal of *unconditional* results. The richness of the space of algorithms is one of the main "excuses" offered for our relatively little success in proving unconditional lower bounds. If indeed this space is much more limited for combinatorial problems, perhaps this can help in finding such proofs.[7]

Acknowledgements. I thank Scott Aaronson, Dimitris Achlioptas, Amin Coja-Oghlan, Tim Gowers, and David Steurer for useful comments and discussion.

[7] In some sense, such an approach to proving lower bounds is dual to Mulumley's approach of "Geometric Complexity Theory" (GCT). (For more information about GCT, see the talks in this workshop, and also this StackExchange answer, this presentation and this paper.) The GCT approach attempts to use specific properties of structured functions such as the permanent to obtain a lower bound; these properties are actually "constructive" in the Razborov-Rudich sense of Natural Proofs. If we focused on combinatorial, "unstructured", problems then we would need to come up with general properties guaranteeing hardness, that would also apply to random functions (which are the ultimate unstructured functions). The Razborov-Rudich result implies such properties would be inherently non-constructive. Valiant's approach for proving certain types of lower bounds via *Matrix Rigidity* can be thought of as an instance of the latter approach.