A FINE-GRAINED APPROACH TO ALGORITHMS AND COMPLEXITY

Virginia Vassilevska Williams
Stanford University
THE CENTRAL QUESTION OF ALGORITHMS RESEARCH

``How fast can we solve fundamental problems, in the worst case?''

etc.
ALGORITHMIC TECHNIQUES

- Divide and Conquer
- Dynamic Programming
- Greedy approaches
- Linear programming
- Color-coding
- Iterative compression
- Semidefinite Programming
- Kernelization
- Inclusion-exclusion
- Backtracking
- Sum of Squares
- Specialized data structures
HARD PROBLEMS

For many problems, the known techniques get stuck:

• Very important computational problems from diverse areas
• They have simple, often brute-force, classical algorithms
• No improvements in many decades!
A CANONICAL HARD PROBLEM

**k-SAT**

*Input*: variables $x_1, \ldots, x_n$ and a formula $F = C_1 \land C_2 \land \ldots \land C_m$ so that each $C_i$ is of the form

$\{y_1 \lor y_2 \lor \ldots \lor y_k\}$ and $\forall i$, $y_i$ is either $x_t$ or $\neg x_t$ for some $t$.

*Output*: A boolean assignment to $\{x_1, \ldots, x_n\}$ that satisfies all the clauses, or NO if the formula is not satisfiable.

Brute-force algorithm: try all $2^n$ assignments.

Best known algorithm: $O(2^{n-(cn/k)}n^d)$ time for const $c, d$.

Goes to $2^n$ as $k$ grows.
ANOTHER HARD PROBLEM: LONGEST COMMON SUBSEQUENCE (LCS)

Given two strings on n letters

ATCGGGTTCCCTAAGGG
ATTTGGTACCCTCAAGGG

Find a subsequence of both strings of maximum length.

Applications both in computational biology and in spellcheckers.

Solved daily on huge strings!
(Human genome: $3 \times 10^9$ base pairs.)
I've got data. I want to solve this algorithmic problem but I'm stuck!

Ok, thanks, I feel better that none of my attempts worked. I'll use some heuristics.

I'm sorry, this problem is NP-hard. A fast algorithm for it would resolve a hard problem in CS/math.
I've got data. I want to solve this algorithmic problem but I'm stuck!

But my data size n is huge! Don't you have a faster algorithm?

?!? ... Should I wait? ... Or should I be satisfied with heuristics?

Great news! Your problem is in P. Here's an $O(n^2)$ time algorithm!

Uhm, I don't know... This is already theoretically fast... For some reason I can't come up with a faster algorithm for it right now...
IN THEORETICAL CS, POLYNOMIAL TIME = EFFICIENT/EASY.

This is for a variety of reasons. E.g. composing two efficient algorithms results in an efficient algorithm. Also, model-independence.

However, noone would consider an $O(n^{100})$ time algorithm efficient in practice. If $n$ is huge, then $O(n^2)$ can also be inefficient.
We are stuck on many problems, even just in $O(N^2)$ time.

No $N^{2-\epsilon}$ time algorithms known for:

- Many string matching problems:
  - Edit distance,
  - Sequence local alignment,
  - LCS,
  - Jumbled indexing …

General form: given two sequences of length $n$, how similar are they? All variants can be solved in $O(n^2)$ time by dynamic programming.

ATCGGGTTTCCTTAAGGG
ATTGGTACCTCCAGG
WE ARE STUCK ON MANY PROBLEMS, EVEN JUST IN $O(N^2)$ TIME

No $N^{2-\varepsilon}$ time algorithms known for:

- Many *string matching* problems
- Many problems in *computational geometry*: e.g.
  
> Given $n$ points in the plane, are any three colinear?  

*A very important primitive!*
WE ARE STUCK ON MANY PROBLEMS, EVEN JUST IN $O(N^2)$ TIME

No $N^{2-\varepsilon}$ time algorithms known for:

- Many string matching problems
- Many problems in computational geometry
- Many graph problems in sparse graphs: e.g.

Given an n node, $O(n)$ edge graph, what is its diameter?

Fundamental problem. Even approximation algorithms seem hard!
WE ARE STUCK ON MANY PROBLEMS, EVEN JUST IN $O(N^2)$ TIME

No $N^{2-\epsilon}$ time algorithms known for:

- Many string matching problems
- Many problems in computational geometry
- Many graph problems in sparse graphs
- Many other problems …

Why are we stuck?

Are we stuck because of the same reason?
PLAN

• Traditional hardness in complexity

• A fine-grained approach

• New Developments
COMPUTATIONAL COMPLEXITY

- **PSPACE**
- **NP**
- **P**
- **Logspace**
- **AC0**

Contains the “1000-clique” problem; best runtime: $\Omega(n^{790})$

QBF with $k$ alternations in $2^{n^{\Omega(k)}}$ time

Even $O(n^2)$ time is inefficient

Best SAT alg: $2^n$

QBF gets easier as the # of quantifiers increases...

This traditional complexity class approach says little about runtime!
WHY IS K-SAT HARD?

Theorem [Cook, Karp’72]:
k-SAT is **NP-complete** for all $k \geq 3$.

That is, if there is an algorithm that solves k-SAT instances on $n$ variables in $\text{poly}(n)$ time, then all problems in NP have $\text{poly}(N)$ time solutions, and so $P=NP$.

k-SAT (and all other NP-complete problems) are considered hard because **fast algorithms for them imply fast algorithms for many important problems**.
TIME HIERARCHY THEOREMS

For most natural computational models one can prove:

for any constant $c$, there exist problems solvable in $O(n^c)$ time but not in $O(n^{c-\varepsilon})$ time for any $\varepsilon > 0$.

It is completely unclear how to show that a particular problem in $O(n^c)$ time is not in $O(n^{c-\varepsilon})$ time for any $\varepsilon > 0$.

Unconditional lower bounds seem hard.

In fact, it is not even known if SAT is in linear time!

We instead develop a fine-grained theory of hardness that is conditional and mimics NP-completeness.
PLAN

• Traditional hardness in complexity

• A fine-grained approach

• Fine-grained reductions lead to new algorithms
IDEA: Mimic NP-completeness

1. Identify key hard problems

2. Reduce these to all (?) other problems believed hard

3. Hopefully form equivalence classes

Goal:
understand the landscape of algorithmic problems
CNF SAT IS CONJECTURED TO BE REALLY HARD

Two popular conjectures about SAT on n variables [IPZ01]:

ETH: 3-SAT requires $2^{\delta n}$ time for some constant $\delta > 0$.

SETH: for every $\varepsilon > 0$, there is a k such that k-SAT on n variables, m clauses cannot be solved in $2^{(1-\varepsilon)n \ poly m}$ time.

So we can use k-SAT as our hard problem and ETH or SETH as the conjecture we base hardness on.
Given a set $S$ of $n$ integers, are there $a, b, c \in S$ with $a + b + c = 0$?

**Conjecture:** Orthog. Vectors requires $n^{2-o(1)}$ time.

**[W’04]:** SETH implies this conjecture!

Easy $O(n^2 d)$ time alg
Best known [AWY’15]: $n^2 - \Theta(1 / \log (d/\log n))$

**Conjecture:** APSP requires $n^{3-o(1)}$ time.

All pairs shortest paths:
given an $n$-node weighted graph, find the distance between every two nodes.

Classical algs: $O(n^3)$ time
[W’14]: $n^3 / \exp(\sqrt{\log n})$ time

Recent research [CGIMPS’16] suggests these problems are not equivalent!

Orthogonal vectors

Given a set $S$ of $n$ vectors in $\{0,1\}^d$, for $d = \omega(\log n)$ are there $u, v \in S$ with $u \cdot v = 0$?

3SUM

Easy $O(n^2)$ time alg
[BDP’05]: $\sim n^2 / \log^2 n$ time for integers
[GP’14]: $\sim n^2 / \log n$ time for reals

More key problems to blame

**Conjecture:** 3SUM requires $n^{2-o(1)}$ time.

**Conjecture:** APSP requires $n^{3-o(1)}$ time.
<table>
<thead>
<tr>
<th>Author</th>
<th>Runtime</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floyd, Warshall</td>
<td>$n^3$</td>
<td>1962</td>
</tr>
<tr>
<td>Fredman</td>
<td>$n^3$</td>
<td>1976</td>
</tr>
<tr>
<td>Takaoka</td>
<td>$n^3$</td>
<td>1992</td>
</tr>
<tr>
<td>Dobosiewicz</td>
<td>$n^3$</td>
<td>1992</td>
</tr>
<tr>
<td>Han</td>
<td>$n^3$</td>
<td>2004</td>
</tr>
<tr>
<td>Takaoka</td>
<td>$n^3$</td>
<td>2004</td>
</tr>
<tr>
<td>Zwick</td>
<td>$n^3$</td>
<td>2004</td>
</tr>
<tr>
<td>Chan</td>
<td>$n^3$</td>
<td>2005</td>
</tr>
<tr>
<td>Han</td>
<td>$n^3$</td>
<td>2006</td>
</tr>
<tr>
<td>Chan</td>
<td>$n^3$</td>
<td>2007</td>
</tr>
<tr>
<td>Han, Takaoka</td>
<td>$n^3$</td>
<td>2012</td>
</tr>
<tr>
<td>Williams</td>
<td>$n^3$</td>
<td>2014</td>
</tr>
</tbody>
</table>

Classical problem
Long history
FINE-GRAINED HARDNESS

Idea: **Mimic NP-completeness**

1. Identify **key hard problems**

2. **Reduce** these to all (?) other hard problems

3. Hopefully form **equivalence classes**

**Goal:**

*understand the landscape of algorithmic problems*
A is \((a,b)\)-reducible to B if for every \(\varepsilon > 0\) \(\exists \delta > 0\), and an \(O(a(n)^{1-\delta})\) time algorithm that adaptively transforms any A-instance of size \(n\) to B-instances of size \(n_1, \ldots, n_k\) so that \(\sum_i b(n_i)^{1-\varepsilon} < a(n)^{1-\delta}\).

**Intuition:** \(a(n), b(n)\) are the naive runtimes for A and B. A reducible to B implies that beating the naive runtime for B implies also beating the naive runtime for A.

- If B is in \(O(b(n)^{1-\varepsilon})\) time, then A is in \(O(a(n)^{1-\delta})\) time.
- Focus on exponents.
- We can build equivalences.

Next: an example
AN EXAMPLE FINE-GRAINED EQUIVALENCE

**THEOREM [VW’10]:** Boolean matrix multiplication (BMM) is equivalent to Triangle detection under *subcubic* fine-grained reductions.

**BMM:** Given two $n \times n$ Boolean matrices $X$ and $Y$, return an $n \times n$ matrix $Z$ where for all $i$ and $j$, $Z[i, j] = \text{OR}_k (X[i, k] \text{ AND } Y[k, j])$.

**Triangle detection:** Given an $n$ node graph $G$, does it contain three vertices $a, b, c$, such that $(a, b), (b, c), (c, a)$ are all edges?

We will show that
(1) an $O(n^{3-e})$ time alg for BMM can give an $O(n^{3-e})$ time triangle alg, and
(2) an $O(n^{3-e})$ time alg for triangle can give an $O(n^{3-e/3})$ time BMM alg.
BMM: Given two n x n Boolean matrices X and Y, return an n x n matrix Z where for all i and j, 
\[ Z[i, j] = \text{OR}_k (X[i, k] \text{ AND } Y[k, j]). \]

G = (V, E) - n node graph. A - n x n adjacency matrix: for all pairs of nodes u, v
\[ A[u, v] = 1 \text{ if } (u, v) \text{ is an edge and } 0 \text{ otherwise.} \]

Say \( Z = \text{Boolean product of } A \text{ with itself}. \) Then for all pairs of nodes u ≠ w,
\[ Z[u, w] = \text{OR}_v (A[u, v] \text{ AND } A[v, w]) = \begin{cases} 1 \text{ if there is a path of length 2 from } u \text{ to } w. \\ 0 \text{ otherwise.} \end{cases} \]

So G has a triangle iff there is some edge (u, w) in G s.t. \( Z[u, w] = 1. \)

If one can multiply Boolean matrices in \( O(n^c) \) time, then one can find a triangle in a graph in \( O(n^c) \) time.
**BMM:** Given two $n \times n$ Boolean matrices $X$ and $Y$, return an $n \times n$ matrix $Z$ where for all $i$ and $j$,

$$Z[i, j] = \bigvee_k (X[i, k] \land Y[k, j]).$$

**Reduction from BMM to triangle finding:**

- Split $A$ into pieces $A_1, \ldots, A_t$ of size $n/t$.
- Split $B$ into pieces $B_1, \ldots, B_t$ of size $n/t$.
- Split $C$ into pieces $C_1, \ldots, C_t$ of size $n/t$.
- Place an edge between every $i$ in $A$ and every $j$ in $C$.
- $Z$ – all zeros matrix.
- For all triples $A_p, B_q, C_r$ in turn:
  - While $A_p \mid B_q \mid C_r$ has a triangle,
    - Let $(i, j, k)$ be a triangle in $A_p \mid B_q \mid C_r$.
    - Set $Z[i, j] = 1$.
    - Remove $(i, j)$ from the graph.
BMM TO TRIANGLE REDUCTION

Correctness: Every triple of nodes $i, j, k$ appears in some examined $A_p[B_q[C_r]$

Runtime: Every call to the Triangle finding algorithm is due to either
(1) Setting an entry $Z[i, j]$ to 1, or

this happens at most once per pair $i, j$

(2) Determining that some triple $A_p[B_q[C_r]$ doesn’t have any more triangles

this happens at most once per triple $A_p[B_q[C_r$

If the runtime for detecting a triangle is $T(n) = O(n^{3-\epsilon})$, then the reduction time is

$$(n^2 + t^3) T(n/t).$$

Setting $t=n^{2/3}$, we get: $O(n^3 - \epsilon/3)$.
FINE-GRAINED HARDNESS

Idea: Mimic NP-completeness

1. Identify key hard problems

2. Reduce these to all (?) other hard problems

3. Hopefully form equivalence classes

Goal:
understand the landscape of algorithmic problems
Using other hardness assumptions, one can unravel even more structure

\[ N \] – input size
\[ n \] – number of variables or vertices

**SOME STRUCTURE WITHIN P**

Sparse graph diameter [RV’13], approximate graph eccentricities [AVW’16], local alignment, longest common substring* [AVW’14], Frechet distance [Br’14], Edit distance [Bl’15], LCS, Dynamic time warping [ABV’15, BrK’15], subtree isomorphism [ABHVZ’15], …

In dense graphs: radius, median, betweenness centrality [AGV’15], negative triangle, second shortest path, replacement paths, shortest cycle … [VW’10], …

Huge literature in comp. geom. [GO’95, BHP98, …]: Geombase, 3PointsLine, 3LinesPoint, Polygonal Containment …

String problems: Sequence local alignment [AVW’14], jumbled indexing [ACLL’14]
PLAN

• Traditional hardness in complexity

• A fine-grained approach

• New developments
  • The quest for more believable conjectures
THE QUEST FOR MORE PLAUSIBLE CONJECTURES

• Two problems harder than CNF-SAT, 3SUM, and APSP

• Longest common subsequence, Formula SAT and Branching Programs
Given an n-node graph G, a color for every vertex in G, and an integer D, is there a triple of colors q1,q2,q3 such that there are at least D triangles in G with node colors exactly q1,q2,q3?

Given an n-node graph G and a color for every vertex in G, is there a triple of colors q1,q2,q3 such that there are no triangles in G with node colors exactly q1,q2,q3?

1. Graphs don’t have weights, just node colors
2. Any reduction from these problems would imply hardness under all three conjectures!

[Abboud-VW-Yu STOC’15]: Two hard problems for node-colored graphs
SOME STRUCTURE WITH

Sparse graph diameter [RV’13], local alignment, longest common substring* [AVW’14], Frechet distance [Br’14], Edit distance [BI’15], LCS, Dynamic Time Warping [ABV’15, BrK’15]…

In dense graphs: radius, median, betweenness centrality [AGV’15], negative triangle, second shortest path, replacement paths, shortest cycle … [VW’10], …

Huge literature in comp. geom. [GO’95, BH-P98, …]: Geombase, 3PointsLine, 3LinesPoint, Polygonal Containment …

String problems: Sequence local alignment [AVW’14], jumbled indexing [ACLL’14]

Dynamic problems [P’10], [AV’14], [HKNS’15], [RZ’04]
THE QUEST FOR MORE PLAUSIBLE CONJECTURES

• Two problems harder than CNF-SAT, 3SUM, and APSP

• Longest common subsequence, Formula SAT and Branching Programs
The most successful hypothesis has been SETH.
It is ultimately about SAT of linear size CNF-formulas.
There are more difficult satisfiability problems:

- **CIRCUIT-SAT**
- **NC-SAT**
- **NC1-SAT** ...

[Circuits](#)

**C-SETH**: satisfiability of circuits from circuit class C on n variables and size s requires $2^{n-o(n)}\text{poly}(s)$ time.

[Williams’ 10, ’11]: a $2^n/n^{10}$ time SAT algorithm implies circuit lower bounds for C (for E$^\text{NP}$ and others); the bigger the class the stronger the lower bound.
A VERY RECENT DEVELOPMENT

**Theorem** [Abboud-Hansen-VW-Williams’16]: There is an efficient reduction from **Satisfiability for non-deterministic branching programs** (BPs) of size $T$ and width $W$ and $n$ input variables to the following string problems on strings of length $N = 2^{n/2} T^{O(\log W)}$:

- Longest Common Subsequence, Edit Distance, Dynamic Time Warping, etc.
A type of reachability question. Proof encodes a Savitch-like construction into the LCS/Edit distance instance.

BP: edge-labelled, directed, layered graph. **Start** node s, **accept** node t. **Width**: W nodes per layer. **Size**: T layers.

Each layer labeled with a variable. A variable can label many layers. Each edge labeled with 0 or 1.

An input 001 is accepted if it generates an s-t path.
A VERY RECENT DEVELOPMENT

**Theorem** [Abboud-Hansen-VW-Williams’16]: There is an efficient reduction from **Satisfiability for non-deterministic branching programs** (BPs) of size $T$ and width $W$ and $n$ input variables to the following string problems on strings of length $N = 2^{n/2} T^{O(\log W)}$:

- Longest Common Subsequence, Edit Distance, Dynamic Time Warping, etc.

[Barrington’85]: BPs with $T=2^{\text{polylog } n}$ and $W=5$ capture NC. The above problems require $N^{2-o(1)}$ time under **NC-SETH**.
MORE CONSEQUENCES OF
“BP-SAT FOR N,T,W → EDIT DISTANCE ETC. ON 2^{N/2} T^{O(\log W)}”

BPs with $T=2^{o(\sqrt{n})}$ and $W=2^{o(\sqrt{n})}$ can represent any non-
deterministic Turing machine using $o(\sqrt{n})$ space

• Edit Distance (or LCS etc) in $O(n^{2-\epsilon})$ time implies a nontrivial improvement over exhaustive search for checking SAT of complex objects that can easily implement e.g. cryptographic primitives

• Much more surprising than refuting SETH!
If Edit Distance (or LCS etc) has $O(n^{2-\varepsilon})$ time algorithms for any $\varepsilon > 0$, then $\mathsf{E^{NP}}$ does not have:

- **Non-uniform $2^{o(n)}$-size Boolean Formulas**
  we don’t even know if the enormous $\Sigma_2^{\exp}$ has $2^{o(n)}$-size depth-3 circuits

- **Non-uniform $o(n)$-depth circuits of bounded fan-in**

- **Non-uniform $2^{o(\sqrt{n})}$-size non-deterministic branching programs**
[Williams’14, Abboud-Williams-Yu’15]: APSP can be solved in $n^3 / \log^{\omega(1)} n$ time, OV can be solved in $n^2 / \log^{\omega(1)} n$ time.

Does Edit Distance (or LCS etc) have such an algorithm? (The current best algorithms run in $\sim n^2 / \log^2 n$ time.)

**PARTIAL ANSWER:** An $n^2 / \log^{\omega(1)} n$ algorithm for Edit Distance (or LCS etc) implies that $E^{NP}$ is not in $NC^1$.

Also meaningful for particular polylogs. E.g. if Edit Distance (or LCS etc) has an $n^2 / \log^{100} n$ time algorithm, then $E^{NP}$ does not have non-uniform Boolean formulas of size $n^5$. 
Thank you!