A FINE-GRAINED APPROACH TO ALGORITHMS AND COMPLEXITY

Virginia Vassilevska Williams
MIT

ICDT 2018
THE CENTRAL QUESTION OF ALGORITHMS RESEARCH

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

etc.
For many problems, the known techniques get stuck:

• Very important computational problems from diverse areas
• They have simple, often brute-force, textbook algorithms…
• …That are slow.
• 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 $\overline{x_t}$ 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.
Given two strings on n letters

\[
\begin{align*}
\text{ATCGGGTTCCATT} & \quad \text{ATAAGGG} \\
\text{ATGGTACCTCAGGG} & \quad \text{ATGGTACCTCAGGG}
\end{align*}
\]

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.)
ANOTHER HARD PROBLEM: BATCH PARTIAL MATCH

Given a database $D \subseteq \{0, 1\}^n$ of size $n$ and queries $x_1, \ldots, x_n \in \{0, 1, *\}^d$, for every $x_i$, report whether there is a $y$ in $D$ that matches $x_i$ in all non-$*$ positions, i.e. for every $c$, either $x_i[c] = *$ or $x_i[c] = y[c]$.  

Example 3

No $O(n^{2-\epsilon})$ time alg. for $\epsilon > 0$ known for $d \geq \omega(\log n)$!

Try all pairs $(y \in D, \text{query } x_i)$ and check whether they match in $O(n^2d)$ time.

Or: Preprocess all possible queries: $O(2^d n)$ time.

[AWY'15]: If $d = c(n) \log n$, $O(n^{2-1/O(\log(c))})$ time.
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, no one would consider an $O(n^{100})$ time algorithm efficient in practice.

If $n$ is huge, then $O(n^2)$ is also inefficient.
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:
  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.

ATCGGGTTCTTAAGGG
ATTGGTACCTTCAGG
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 collinear?
  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 $N^2 - \varepsilon$ time algorithms known for:

- Many *string matching* problems
- Many problems in *computational geometry*
- Many *graph problems* in sparse graphs
- Many problems from *databases*:
  - batch partial match, $Q = R(A, B) \Join S(B, C) \Join T(C, A) /$ triangle listing
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
- Many problems from databases
- 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
This traditional complexity class approach says little about runtime.

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

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

Best SAT alg: $2^n$

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

QBF with $k$ alternations in $2^n - \Omega(k)$ time [SW15]
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 $\varepsilon > 0$.

*It is not even known if SAT is in linear time!*
Why is k-SAT hard?

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

I.e. k-SAT is considered hard because "\textit{fast}" algorithms for it imply "\textit{fast}" algorithms for many important problems.

We’ll develop a \textit{fine-grained theory of hardness} that is conditional and mimics NP-completeness.
PLAN

• Traditional hardness in complexity

• A fine-grained approach

• Examples
FINE-GRAINED HARDNESS

Idea: Mimic NP-completeness

1. Identify key hard problems

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

3. Hopefully form equivalence classes
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 $\epsilon > 0$, there is a $k$ such that $k$-SAT on $n$ variables, $m$ clauses cannot be solved in $2^{(1-\epsilon)n \ poly m}$ time.

So we can use $k$-SAT as our hard problem and ETH or SETH as the hypothesis we base hardness on.
**3SUM**
Given a set $S$ of $n$ integers, are there $a, b, c \in S$ with $a + b + c = 0$?

Easy $O(n^2)$ time alg

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

**Batch Partial Match (BPM)**
Given a database $D$ of $n$ vectors in $\{0,1\}^d$ and $n$ queries $x_1, \ldots, x_n$ in $\{0,1,*\}^d$ for $d = \omega(\log n)$, for every $x_i$, answer whether some $y$ in $D$ matches it.

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

Hypothesis: BPM requires $n^{2-o(1)}$ time.

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

Classical alg: $O(n^3)$ time

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

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

**Orthogonal Vectors...**

**Fix the model:** word-RAM with $O(\log n)$ bit words

**Hypothesis:** BPM requires $n^{2-o(1)}$ time.

[W’05]: SETH implies this hypothesis!

Strengthening of SETH [CGIMPS’16] suggests these are **not equivalent**...

More key problems to blame

Classical alg: $O(n^3)$ time
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**
• 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 \(\sum_i b(n_i)^{1-\varepsilon} < a(n)^{1-\delta}\).

- 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.

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.
Example: Batch Partial Match ~ Exists Partial Match

Input: \( D \subseteq \{0, 1\}^d, |D| = n, x_1, \ldots, x_n \in \{0, 1, \star\}^d, d = \omega(\log n) \)

BPM Output: For every \( x_i \), is there a \( y \in D \) matching \( x_i \)?

EPM Output: Is there an \( x_i \) and a \( y \in D \) matching \( x_i \)?

Theorem (VW-W'10):

If EPM on N size inputs is in \( O(N^{2-\epsilon}) \) time, then BPM is in \( O(n^{2-\epsilon/2}) \) time.
**Input:** \( D, X = \{x_1, ..., x_n\} \)

BPM Output: for each \( x_i \), is there a \( y \in D \) that matches \( x_i \)?

EPM Output: Is there any \( x_i \) with a \( y \in D \) that matches it?

**Reduction from BPM to EPM:** \( t = n^{1/2} \)

- Split \( D \) into pieces \( D_1, ..., D_t \) of size \( n/t \)
- Split \( X \) into pieces \( X_1, ..., X_t \) of size \( n/t \)
- \( Z \) – all zeros vector
- For all **pairs** \((D_p, X_q)\) in turn:
  - While EPM finds a partial match \((y \in D, x_i \in X)\) for \((D_p, X_q)\):
    - Set \( Z[i] = 1 \)
  - **Remove** \( x_i \) from \( X \).
Z – all zeros vector
For all pairs \((D_p, X_q)\) in turn:
  While EPM finds a partial match 
  \((y \in D, x_i \in X)\) for \((D_p,X_q)\):
    Set \(Z[i] = 1\)
    Remove \(x_i\) from \(X\).

**Correctness:** Every pair \((y \in D, x_i \in X)\) appears in some examined \(D_p, X_q\)

**Runtime:** Every call to the Exists Partial Match finding algorithm is due to either
  (1) Setting an entry \(Z[i]\) to 1, or

  this happens at most once per \(x_i \in X\).

  (2) Determining that some pair \((D_p, X_q)\) doesn’t have any more partial matches

  this happens at most once per pair \((D_p, X_q)\),

  If the runtime for detecting a partial match is \(T(n) = O(n^{2-\epsilon})\), then the reduction time is

  \((n + t^2) T(n/t)\). Setting \(t=n^{1/2}\), we get: \(O(n^{2 - \epsilon/2})\).
Using other hardness assumptions, one can unravel even more structure

N – input size
n – number of variables or vertices

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

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

Graph diameter [RV’13, BRSVW’18], eccentricities [AVW’16], local alignment, longest common substring* [AVW’14], Frechet distance [Br’14], Edit distance [Bl’15], LCS, Dyn. time warping [ABV’15, BrK’15], subtree isomorphism [ABHVZ’15], Betweenness [AGV’15], Hamming Closest Pair [AW15], Reg. Expr. Matching [Bl16, BGL17] …

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

Many dynamic problems [P’10, AV’14, HKNS’15, D16, RZ’04, AD’16] …

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

• Traditional hardness in complexity

• A fine-grained approach

• More examples
LISTING TRIANGLES

Suppose that $G = (V, E)$ is an $m$ edge graph and we want to list its triangles. How fast can we do it?

Corresponds to the query $R(A, B) \bowtie S(B, C) \bowtie T(C, A)$ where $R, S, T$ have $m$ tuples.

$G$ can have $\sim m^{3/2}$ triangles, and these can be listed in $\sim m^{3/2}$ time.

What if $G$ has $\sim m$ triangles?

Patrascu'10: under the 3SUM Hypothesis, $m^{\frac{4}{3} - o(1)}$ time is needed!

Bjorklund et al'14: if $n \times n$ matrices can be multiplied in $O(n^2)$ time (huge open problem), then can list $m$ triangles in $\sim m^{4/3}$ time.

Follow-up work – Lincoln et al.'17: finding directed cycles is hard…
The fine-grained approach has been spreading:

- Space usage of data structures [GKLP’17]
- Planar graph problems [AD’16]
- Sparse graph problems [LVW’18]
- Fine-grained Fixed Parameter Tractability [AVW’16]
- Fine-grained Cryptography [BRSV’17]
- Fine-grained I/O Complexity [DLLLV’18]
- Time/space tradeoffs [LVW’16]
- Approximation hardness
- More-believable hypotheses
Thank you!

Questions?

Many open problems: bit.ly/2mdi6zH

New survey: bit.ly/2D9jbAx