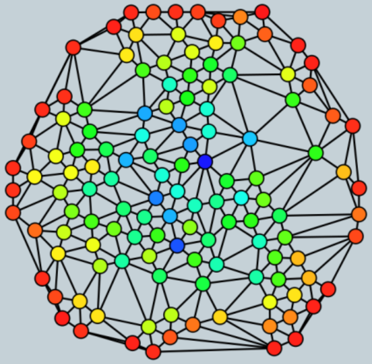
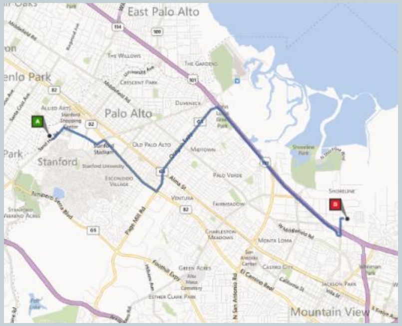
Hardness for Easy Problems

VIRGINIA V. WILLIAMS

STANFORD CS DEPT.





What is a hard computational problem?

k-SAT

Input: variables $x_1, \ldots, x_n$ and a formula

$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ so that each C_i is of the form

$\{y_1 \vee y_2 \vee \ldots \vee y_k\}$ and $\forall j$, y_j 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

Example: $\{\neg x_1 \vee x_2\} \wedge \{x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5\} \wedge \{\neg x_4 \vee \neg x_2\}$

What is a hard computational problem?

k-SAT

Input: variables $x_1, \ldots, x_n$ and a formula

$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ so that each C_i is of the form

$\{y_1 \vee y_2 \vee \ldots \vee y_k\}$ and $\forall j$, y_j 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

Trivial algorithm: try all 2^n assignments

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

Why is k-SAT hard?

Theorem [Cook,Karp'72]:

k-SAT is **NP-complete** for all k≥3.

That is, if there is an algorithm that solves k-SAT instances on n variables in poly(n) time, then all problems in NP have 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 algs for many important problems***.

Polytime solvable means easy?

In theoretical CS, polynomial time = efficient.

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.

What about $O(n^3)$ or $O(n^2)$?

If n is huge, then these can also be inefficient.

Hard poly-time solvable problems

There are many problems that have polynomial time algorithms but no known *really efficient* algorithms.

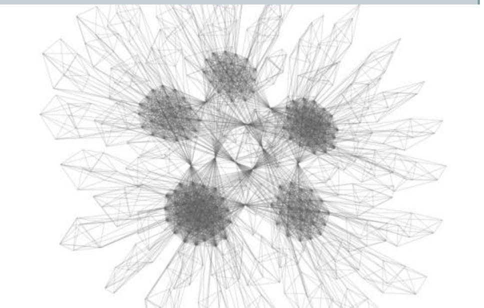
Simple examples:

Many parametrizations of NP-hard problems:

E.g. find a clique on 99 nodes in an n-node graph.

Best running time is $O(n^{79})$...

But there are also some natural problems in $O(n^2)$ time with no practical algorithms



Sequence local alignment

A problem from computational biology:
Given two DNA strings

ATCGGGTTCCTTAAGGG
ATTGGTACCTTCAGG

How similar are they? What do they have in common?

Sequence local alignment

A problem from computational biology:
Given two DNA strings

ATCGGGTTCCTTAAGGG
ATTGG_TACCTTCA_GG

How similar are they? What do they have in common?

Say we are given a score matrix that gives a score for each match (AA,CC,TT,GG), mismatch, or each gap.

Find substring of largest score.
Solved daily on huge strings!!

Longest substring with don't cares

Two strings, one over {0,1}, one over {0,1,*}

000111110001

0**101*110110

Find the longest string that is a substring of both.

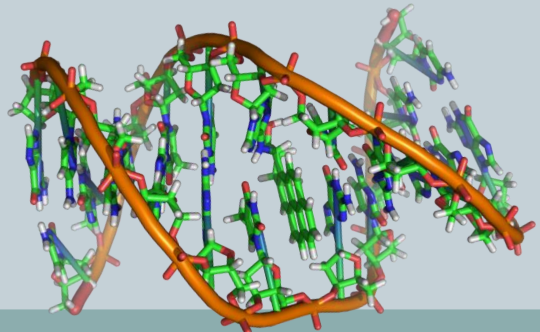
* can be interpreted as 0 or 1.

Sequence problems theory/practice

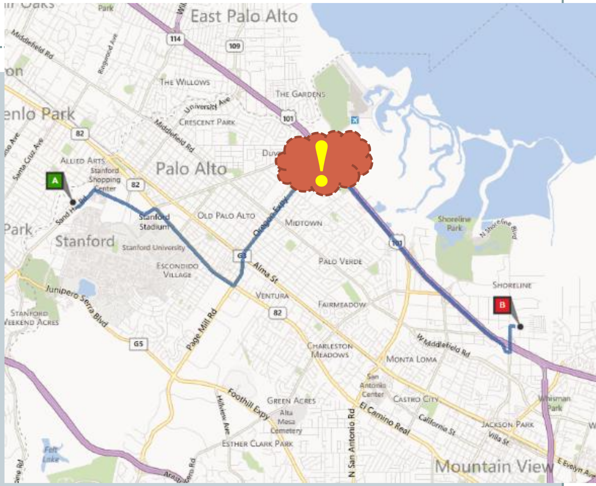
Fastest algorithm: $O(n^2)$ time on length n sequences

Sequence alignment is run on whole genome sequences. Human genome: 3 x 10^9 base pairs.

A quadratic time algorithm is not fast!



Other hard polynomial time problems



Shortest path:

Network *G*, two fixed nodes *s* and *t*

What is the *distance* between s and t?

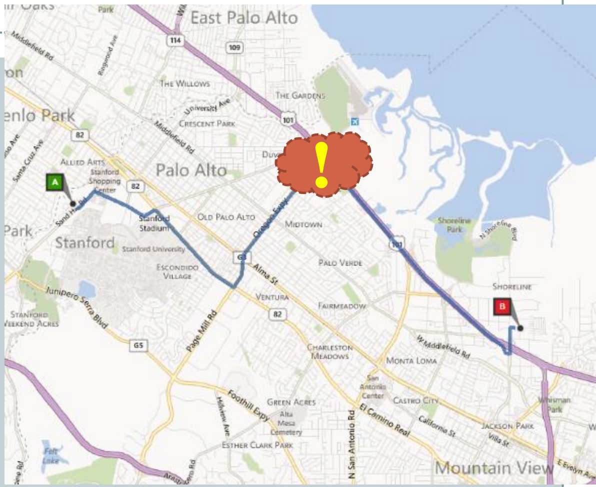
Fast solutions: Dijkstra's algorithm

O(m+n log n) time on m-edge, n-node graph

What if G keeps changing: links are going down and up?

Do we have to recompute the path *from scratch* after each change?

Other hard polynomial time problems

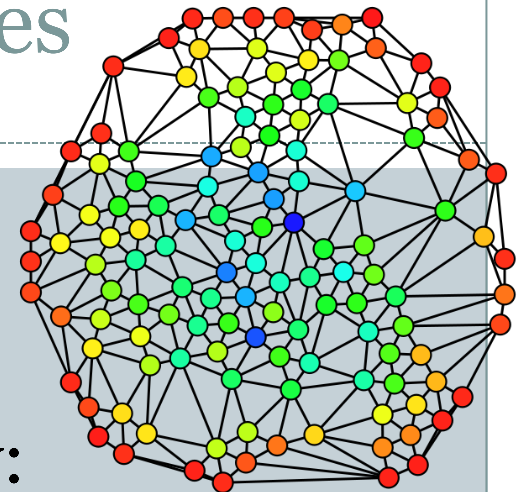


Dynamic s-t shortest path:
Given a graph and fixed vertices s and t, support updates: insert/delete an edge, answer queries “what is d(s,t)?”

Trivial solution: recompute d(s,t) after each update in ~m time.

The brute-force recomputation is the best known!!!

Graph centrality measures



In network analysis, we want to know how *important* nodes are.

Various notions of "centrality" of a node v:

- **Closeness centrality:** $1 / (\Sigma_x d(v,x))$
  - Median: node of largest closeness centrality
- **Betweenness centrality:**

  $\Sigma_{x,y}$ [#shortest xy paths through v]/[#shortest xy paths]
- **Center** of a graph: node c minimizing $\max_u d(c,u)$

All these measures can be computed in $\sim n^{3-o(1)}$ time, and no better algorithms are known.

Some polytime problems seem hard

- Sequence local alignment, dynamic st shortest paths, graph centrality are all important problems with easy, sometimes brute-force, polynomial solutions, but not practical

- Best algorithm for each of them is longstanding, no real improvements for decades

How do we explain this?

Hardness for easy problems

- Let's say that a problem in P is n^c-hard if an $O(n^{c-\varepsilon})$ time algorithm for it for $\varepsilon > 0$, would imply surprising improvements for many famous problems

- What are some famous hard problems?

(1) Is k-SAT in 1.9^n time for all k?

Faster k-SAT?

The fastest algs for k-SAT run in $O(2^{n-cn/k})$ time.

This is essentially 2^n for large k!

Huge open problem to improve on this.

Strong Exponential Time Hypothesis

SETH [IPZ'01]: For every $\varepsilon>0$, there is a k such that k-SAT is not in $O((2-\varepsilon)^n)$ time.

Widely believed. Many conditional results.

Hardness for easy problems

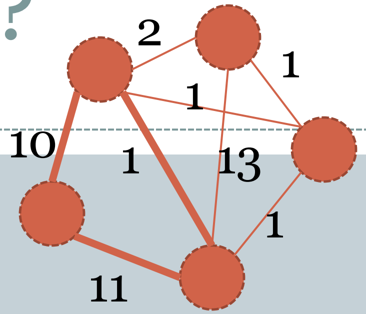
- Let's say that a problem in P is n^c-hard if an $O(n^{c-\varepsilon})$ time algorithm for it for $\varepsilon > 0$, would imply surprising improvements for many famous problems

- What are some famous hard problems?

(1) Is k-SAT in 1.9^n time for all k?

(2) Is weighted k-clique in $n^{0.9k}$ time?

Faster weighted k-clique?



Weighted k-clique

Input: a graph with weights on edges

Output: k nodes forming a clique, maximizing the sum of the edge weights between the nodes

Trivial algorithm: ~n^k time.

This is essentially the best known!

Weighted Clique Conjecture: No $O(n^{(1-\varepsilon)k})$ time algorithm for weighted k-clique for any $\varepsilon > 0$.

Hardness for easy problems

- Let's say that a problem in P is n^c-hard if an $O(n^{c-\varepsilon})$ time algorithm for it for $\varepsilon > 0$, would imply surprising improvements for many famous problems

- What are some famous hard problems?

(1) Is k-SAT in 1.9^n time for all k?

(2) Is weighted k-clique in $n^{0.9k}$ time?

(3) Is 3SUM in $n^{1.9}$ time?

Faster 3-SUM?

3-SUM
Input: n integers,
Output: Do 3 of the integers sum to 0?

Important problem in computational geometry.
Many problems are equivalent to it. For instance:

Given n points in the plane, are there 3 that are collinear?

Easy n^2 time solution. Best alg: $O(n^2/\log^2 n)$

3SUM conjecture:
There is no $O(n^{2-\varepsilon})$ time alg for 3SUM for any $\varepsilon > 0$.

Hardness for easy problems

- Let's say that a problem in P is n^c-hard if an $O(n^{c-\varepsilon})$ time algorithm for it for $\varepsilon > 0$, would imply surprising improvements for many famous problems

- What are some famous hard problems?

(1) Is k-SAT in 1.9^n time for all k?
(2) Is weighted k-clique in $n^{0.9k}$ time?
(3) Is 3SUM in $n^{1.9}$ time?
(4) Is APSP in $n^{2.9}$ time?

Faster APSP?

All-pairs shortest paths (APSP):

Input: Graph on n nodes

Output: the distances between all pairs of vertices

$O(n^3)$ time algorithm by Floyd-Warshall from 1950s

Current best: Williams'14, $n^3/c^{\sqrt{\log n/\log\log n}} \sim n^{3-o(1)}$

APSP Conjecture:

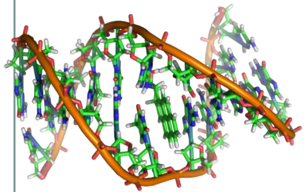
No $O(n^{3-\varepsilon})$ time alg for APSP for any $\varepsilon > 0$.

Hardness for easy problems

- Let's say that a problem in P is n^c-hard if an $O(n^{c-\varepsilon})$ time algorithm for it for $\varepsilon > 0$, would imply surprising improvements for many famous problems

- What are some famous hard problems?

(1) Is k-SAT in 1.9^n time for all k?
(2) Is weighted k-clique in $n^{0.9k}$ time?
(3) Is 3SUM in $n^{1.9}$ time?
(4) Is APSP in $n^{2.9}$ time?
(5) …



Example 1: Sequence alignment

ATTC
GTCC
00111
1*111

Theorem [AVW'14]: If the *Longest substring with don't cares* on n length strings is in $O(n^{1.9})$ time, then k-SAT is in $O(1.99^n)$ time for all k.

So any nontrivial solution implies SETH is false!

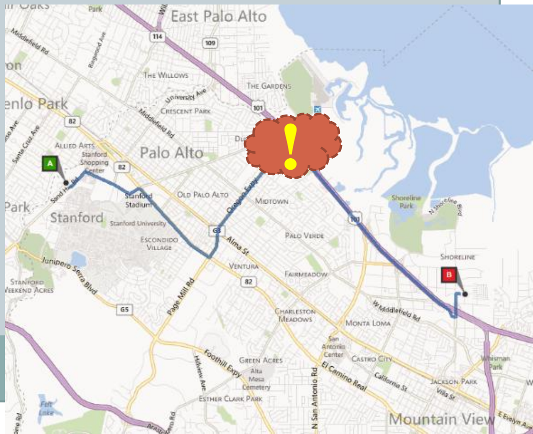
Theorem [AVW'14] If *Local alignment* of sequences of length n is in $O(n^{1.9})$ time, then

- k-SAT is in $O(1.99^n)$ time
- 3-SUM is in $O(n^{1.99})$ time
- Weighted 4-clique is in $O(n^{3.99})$ time

Example 2: Dynamic st-shortest paths

Theorem [AV'14]: If updates can be supported in $O(m^{0.9})$ time, then APSP is in $O(n^{2.99})$ time.

Any nontrivial distance update algorithm would break the APSP conjecture!



Example 3: Graph Centrality Measures

Theorem [AGV'14]:

If the center or median of a graph, or the betweenness centrality of a given vertex can be computed in $O(n^{2.9})$ time, then APSP is in $O(n^{2.99})$ time.

In fact, all these centrality measures are *equivalent* to APSP!

Any subcubic algorithm for one of them implies a subcubic algorithm for all of them.

How do we prove such results?

- With NP-hardness we rely on poly time reductions:

Instance of problem Q $\xrightarrow{\text{alg}}$ instance of problem Q'

- For the type of results we want, this type of reduction can't work.

  E.g., we won't be able to reduce APSP to a problem that asks for a single number, e.g. median.

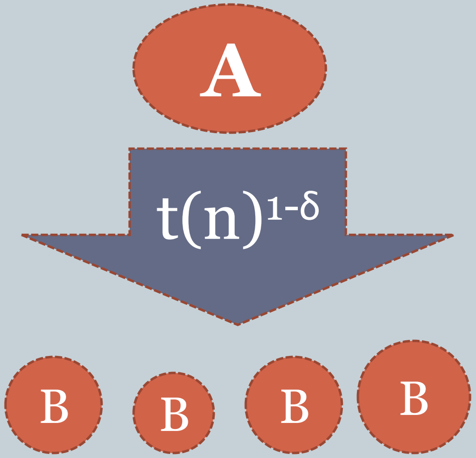
  Also, at least we want the runtime of the reduction to be low… Need more refined reductions.

Fine-grained reductions

- A is reducible to B if

  for every $\varepsilon>0$ $\exists$ $\delta>0$, and an $O(t(n)^{1-\delta})$ time algorithm that transforms any A-instance of size n to B-instances of size $n_1,\ldots,n_k$ so that $\sum_i n_i^{c-\varepsilon} < t(n)^{1-\delta}$.

- **If B is in** $O(n^{c-\varepsilon})$ **time,**
  then A is in $O(t(n)^{1-\delta})$ time.
- Focus on exponents.
- P vs NP: poly time reductions.
- Here: more refined notion.



A theory of hardness for polynomial times.

Fine-grained reductions

- Such reductions can preserve exact running times.
- Example: dynamic st-shortest paths

APSP on n nodes $\xrightarrow{n^{8/3}\text{ time}}$ $n^{7/3}$ instances of dyn st-shortest path on $n^{2/3}$ edges, **$n^{7/3}$ updates**

If update time is $m^{1-\varepsilon}$ for some $\varepsilon>0$, then runtime is

$$n^{8/3} + n^{7/3} \times n^{2/3(1-\varepsilon)} = n^{3-2\varepsilon/3} + n^{8/3} = n^{3-\delta}.$$

Any nontrivial update time falsifies APSP conj!

Discussion

- The reductions explain why it has been so hard to obtain improvements.
- The current state of our algorithmic techniques reaches the same *roadblock* on a lot of problems.
- It could be that they are all hard…
- Does this mean we should give up?

No! The reductions can also be viewed as *opportunities* to look at various longstanding open problems from new viewpoints.

Thank you!

virgi@cs.stanford.edu