How hard is it to find a good solution?

Johan Håstad

KTH Computer Science
and Communication

Simons Institute Open Lecture
November 4, 2013

Research Area: Complexity Theory

Given a computational problem, find an efficient algorithm that solves it.

Goal of lecture:

- Give some examples of results.
- Give some flavor of reasoning.
- Pose some open problems.

I am happy to take questions/comments during the talk.

Efficient computation in practice

Given the time-table of all buses and trains in a large city, what is the fastest way to get from point A to point B?

Need an answer within one second.

Can we break the AES encryption scheme using a 1000 PCs in time less than a year?

Efficient computation in theory

How many elementary operations are needed to solve a problem as a function of input size n?

Example: Multiplication of n-digit numbers.

School book

The case of $n = 5$.

$$\begin{array}{cccccccccc}
 & & & & & 2 & 9 & 1 & 2 & 3 \\
 & & & & * & 5 & 1 & 2 & 3 & 4 \\
\hline
 & & & & 1 & 1 & 6 & 4 & 9 & 2 \\
 & & & & 8 & 7 & 3 & 6 & 9 & \\
 & & & 5 & 8 & 2 & 4 & 6 & & \\
 & & 2 & 9 & 1 & 2 & 3 & & & \\
1 & 4 & 5 & 6 & 1 & 5 & & & & \\
\hline
1 & 4 & 9 & 2 & 0 & 8 & 7 & 7 & 8 & 2
\end{array}$$

Requires about $4n^2$ multiplication and additions of one digit numbers.

Best possible?

Obviously best possible!

Clearly each digit of the first number has to be multiplied by each digit of the second number.

Obviously best possible!

Clearly each digit of the first number has to be multiplied by each digit of the second number.

A completely bogus argument!

A sketch of a fast algorithm

Multiplication is very similar to convolution.

- Think of a number as a vector of digits.
- Do a Fast Fourier Transform (FFT) to each vector (number).
- Multiply transforms point-wise.
- Do an inverse FFT.
- Tidy up the result.

Fast multiplication

Choosing suitable domains to do the FFT, applying recursion to smaller multiplications, etc

Theorem: [Fürer, 2007] Multiplication can be done in slightly more than $n \log n$ bit operations.

This improved a 35 year old running time.

Fast multiplication

Choosing suitable domains to do the FFT, applying recursion to smaller multiplications, etc

Theorem: [Fürer, 2007] Multiplication can be done in slightly more than $n \log n$ bit operations.

This improved a 35 year old running time.

Unknown whether multiplication can be done in $10n$ operations.

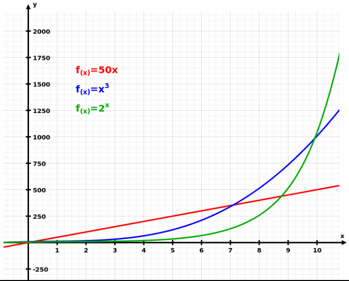
The setting of this talk

Running times which is polynomial in the input size are good.

$$10n, n \log n, 10000n^{100}$$

Running times which grow faster than polynomial in the input size are not good (infeasible computations).

$$2^n, n^{\sqrt{n}}$$



Traveling Salesperson Problem (TSP)

Want to visit the n (100 to be concrete) largest cities in Sweden and return to the start.

Which order to use to minimize total travel distance?

Not so busy salesperson

Five cities, find the best tour

	Stockholm	Göteborg	Luleå	Uppsala	Malmö
Stockholm	0	471	904	69	614
Göteborg	471	0	1250	455	274
Luleå	904	1250	0	836	1472
Uppsala	69	455	836	0	680
Malmö	614	274	1472	680	0

Easy: Try all orderings of the cities.

Shortest tour is 3043 kilometers (S-U-L–G-M-S).

Exhaustive search

Trying all possibilities with $n = 100$. We can fix the starting point, 99 possibilities for first city, 98 for next, etc

In total 99*98*97..*1 possibilities

Exhaustive search

Trying all possibilities with $n = 100$. We can fix the starting point, 99 possibilities for first city, 98 for next, etc

In total 99*98*97..*1 possibilities
$=$933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000

This number is denoted by 99! “factorial”.

Computers are fast

A standard computer makes one billion operations in a second.

Even a stupid program solves TSP with 14 cities in a second, (but does not finish in our life-time with a 100 cities).

Computers are fast

A standard computer makes one billion operations in a second.

Even a stupid program solves TSP with 14 cities in a second, (but does not finish in our life-time with a 100 cities).

Do you expect to live a billion seconds?

Famous quote

Premature optimization is the root of all evil. (Knuth?)

Famous quote

Premature optimization is the root of all evil. (Knuth?)

Complete absence of optimization might be even worse.

Returning to TSP

Dynamic programming can solve TSP in time $n^2 2^n$.

Solves problems with 25 cities in a second but still a 100-city problems takes more than a life-time.

Returning to TSP

Dynamic programming can solve TSP in time $n^2 2^n$.

Solves problems with 25 cities in a second but still a 100-city problems takes more than a life-time.

Best possible?

Not known, not even known to require $10n^2$ operations, i.e. essentially reading the input!

The NP-complete problems

A set containing thousands of computational problems.

None known to be solvable efficiently (polynomial time).

If one is solvable in polynomial time, then so are all.

"NP $=$ P?", is the question whether all these problems can be solved by an efficient algorithm.

NP-complete problems

- Integer programming.
- Almost all scheduling problem.
- Independent set in graphs.
- Graph coloring.
- Knapsack.
- Satisfiability of Boolean formulas.
- Set cover.
- Traveling Salesperson problem

The state of the world

Every reasonable researcher in the area believes that NP $\neq$ P.

We are nowhere close to proving this fact.

We assume NP $\neq$ P and derive consequences of this assumption.

A law of nature.

If this is false as stated hopefully something similar is true.

A different world

In a world where NP really equals P.

- All planning problems are easy.
- To prove a theorem is as easy to verify the proof.
- No security/privacy on the Internet.

TSP, again

TSP is NP-complete and hence we should not expect a fast algorithm that always find the best solution.

Let us turn to heuristics, two basic types:

1. Algorithms that always finds the best solution. Sometimes slow.
2. Algorithms that are always fast. Finds a reasonably good solution.

We focus on the second type.

The first heuristic for TSP

“Greedy” or “Nearest Neighbor”

Go to the closest city not yet visited.

Very efficient to calculate.

Sometimes works well, but not always.

How do we quantify performance?

Measure of goodness

The ratio of found solution to optimal solution, i.e.

$$\frac{\text{Cost of found solution}}{\text{Cost of optimal solution}}$$

To analyze Greedy Heuristic for TSP we need two results, n is the number of cities.

1. A mathematical proof that the algorithm always returns a solution that has cost at most $c \log n \times OPT$.
2. An example of an instance where the answer is as bad as $d \log n \times OPT$.

Jargon

Greedy is a $\Theta(\log n)$-approximation algorithm for TSP.

Can we get within a constant ratio independent of the number of cities?

A classical result

Christofides, 1976. There is an efficient algorithm that always returns a solution for TSP that has cost 1.5 times cost of optimal.

- Find a minimum cost spanning tree.
- Find a minimum cost matching of vertices of odd degree in found spanning tree.
- Make shortcuts.

In pictures

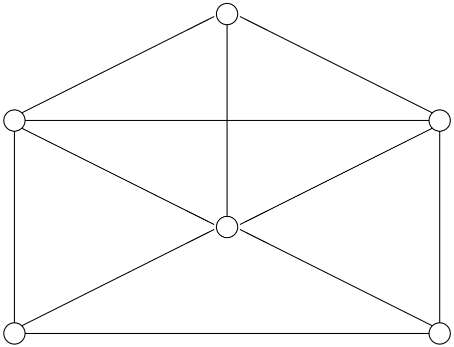


Figure: A graph

In pictures

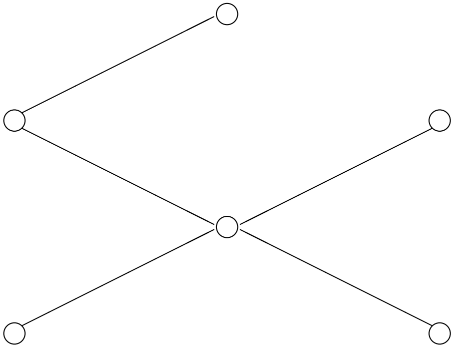


Figure: A spanning tree

In pictures

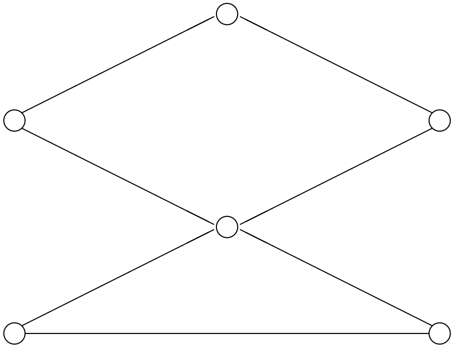


Figure: Adding a matching

In pictures

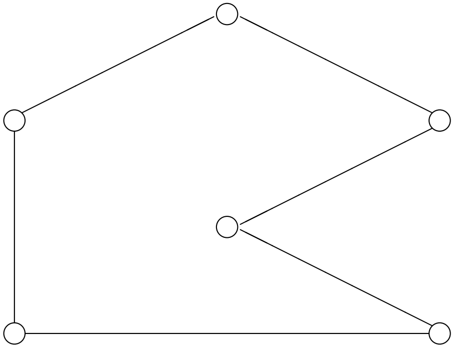


Figure: Taking a shortcut

Homework problem

Find a polynomial time algorithm for TSP that always finds a tour that is at most 1.49 times optimal.

Homework problem

Find a polynomial time algorithm for TSP that always finds a tour that is at most 1.49 times optimal.

Algorithmic researchers have been trying to do this for 37 years.

Please note

Getting an algorithm that beats factor 1.5 on most instances is easy.

If distances are given by distances in the plane it is possible to get arbitrarily close to optimal (Arora, Mitchell, 1998)

Want factor 1.49 only using triangle-inequality:

$d(A, C) \leq d(A, B) + d(B, C)$

Maybe best possible?

Can we prove that 1.5 is best possible?

Need to assume that NP$\neq$P, but we are treating this as a law of nature.

Maybe best possible?

Can we prove that 1.5 is best possible?

Need to assume that NP$\neq$P, but we are treating this as a law of nature.

Theorem: [KLS, 2013]: If NP$\neq$ P, then TSP cannot be approximated within $\frac{123}{122}$ in polynomial time.

Another type of problem

Formulas over Boolean variables.

x_i, Variables that takes values true (T) or false (F).

$\overline{x_i}$ Negations of variables.

$\wedge$ Logical “AND”.

$\vee$ Logical “OR”.

An example

x_1 "Alice drives on the left-hand side going west"

x_2 "Bob drives on the left-hand side going east"

The event "No collision" then becomes

$$(x_1 \wedge x_2) \vee (\overline{x_1} \wedge \overline{x_2})$$

Possible to satisfy, for instance with $x_1 = T$ and $x_2 = T$.

An example

x_1 "Alice drives on the left-hand side going west"

x_2 "Bob drives on the left-hand side going east"

The event "No collision" then becomes

$$(x_1 \wedge x_2) \vee (\overline{x_1} \wedge \overline{x_2})$$

Possible to satisfy, for instance with $x_1 = T$ and $x_2 = T$.

Logically same as

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$$

A bigger formula

$$\begin{array}{ccccc}
(x_1 \vee x_2 \vee x_3) & \wedge & (\overline{x_1} \vee x_2 \vee \overline{x_4}) & \wedge \\
(x_1 \vee x_3 \vee \overline{x_4}) & \wedge & (x_2 \vee \overline{x_3} \vee \overline{x_4}) & \wedge \\
(\overline{x_2} \vee x_3 \vee \overline{x_4}) & \wedge & (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) & \wedge \\
(x_2 \vee x_3 \vee x_5) & \wedge & (\overline{x_2} \vee \overline{x_3} \vee x_5) & \wedge \\
(\overline{x_2} \vee x_4 \vee x_5) & \wedge & (x_3 \vee x_4 \vee \overline{x_5}) & \wedge \\
(\overline{x_3} \vee \overline{x_4} \vee x_5) & \wedge & (\overline{x_3} \vee x_4 \vee \overline{x_5}) &
\end{array}$$

Is it possible to satisfy all the constraints?

A bigger formula

$$\begin{array}{ccccc}
(x_1 \vee x_2 \vee x_3) & \wedge & (\overline{x_1} \vee x_2 \vee \overline{x_4}) & \wedge \\
(x_1 \vee x_3 \vee \overline{x_4}) & \wedge & (x_2 \vee \overline{x_3} \vee \overline{x_4}) & \wedge \\
(\overline{x_2} \vee x_3 \vee \overline{x_4}) & \wedge & (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) & \wedge \\
(x_2 \vee x_3 \vee x_5) & \wedge & (\overline{x_2} \vee \overline{x_3} \vee x_5) & \wedge \\
(\overline{x_2} \vee x_4 \vee x_5) & \wedge & (x_3 \vee x_4 \vee \overline{x_5}) & \wedge \\
(\overline{x_3} \vee \overline{x_4} \vee x_5) & \wedge & (\overline{x_3} \vee x_4 \vee \overline{x_5}) &
\end{array}$$

$$\begin{array}{lcl}
x_1 & = & \text{FALSE} \\
x_2 & = & \text{FALSE} \\
x_3 & = & \text{TRUE} \\
x_4 & = & \text{FALSE} \\
x_5 & = & \text{FALSE}
\end{array}$$

Is it possible to satisfy all the constraints?

Vocabulary

The problem of satisfying Boolean formulas is called "Satisfiability" or simply "Sat".

"Conjunctive Normal Form" (k-CNF), conjunction (AND) of many clauses each the OR of k literals (variables or negations of variables).

Satisfiability of formulas on 3-CNF is called 3-Sat.

3-Sat is NP-complete

3-Sat was essentially the first problem to be proved NP-complete.

Let us look at the problem of satisfying as many constraints as possible.

The set-up

Many constraints, each the OR of three Boolean literals, e.g.

$$\begin{array}{ccccc}
(x_1 \vee x_2 \vee x_3) & \wedge & (\overline{x_1} \vee x_2 \vee \overline{x_4}) & \wedge \\
(x_1 \vee x_3 \vee \overline{x_4}) & \wedge & (x_2 \vee \overline{x_3} \vee \overline{x_4}) & \wedge \\
(\overline{x_2} \vee x_3 \vee \overline{x_4}) & \wedge & (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) & \wedge \\
(x_2 \vee x_3 \vee x_5) & \wedge & (\overline{x_2} \vee \overline{x_3} \vee x_5) & \wedge \\
(\overline{x_2} \vee x_4 \vee x_5) & \wedge & (x_3 \vee x_4 \vee \overline{x_5}) & \wedge \\
(\overline{x_3} \vee \overline{x_4} \vee x_5) & \wedge & (\overline{x_3} \vee x_4 \vee \overline{x_5}) &
\end{array}$$

We are promised that there is a way to satisfy all constraints.

Looking for some efficient way to satisfy as many as possible.

The set-up

Many constraints, each the OR of three Boolean literals, e.g.

$$\begin{array}{ccccc}
(x_1 \vee x_2 \vee x_3) & \wedge & (\overline{x_1} \vee x_2 \vee \overline{x_4}) & \wedge \\
(x_1 \vee x_3 \vee \overline{x_4}) & \wedge & (x_2 \vee \overline{x_3} \vee \overline{x_4}) & \wedge \\
(\overline{x_2} \vee x_3 \vee \overline{x_4}) & \wedge & (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) & \wedge \\
(x_2 \vee x_3 \vee x_5) & \wedge & (\overline{x_2} \vee \overline{x_3} \vee x_5) & \wedge \\
(\overline{x_2} \vee x_4 \vee x_5) & \wedge & (x_3 \vee x_4 \vee \overline{x_5}) & \wedge \\
(\overline{x_3} \vee \overline{x_4} \vee x_5) & \wedge & (\overline{x_3} \vee x_4 \vee \overline{x_5}) &
\end{array}$$

We are promised that there is a way to satisfy all constraints.

Looking for some efficient way to satisfy as many as possible.

Rock bottom amount of thinking

Do not look at the formula.

Set each variable with equal probability to T or F.

Satisfies each constraint with probability $7/8$ and hence gives this approximation ratio.

A surprising theorem

Given a satisfiable formula on 3-CNF, i.e. a collection of constraints on our favorite form.

Theorem: [H, 2001] If NP$\neq$P then it is impossible to efficiently find an assignment that satisfies a fraction .876 of the constraints.

.876 can be replaced by any number greater than $7/8 = .875$.

A surprising theorem

Given a satisfiable formula on 3-CNF, i.e. a collection of constraints on our favorite form.

Theorem: [H, 2001] If NP$\neq$P then it is impossible to efficiently find an assignment that satisfies a fraction .876 of the constraints.

.876 can be replaced by any number greater than $7/8 = .875$.

3-Sat is a too complicated problem for efficient computation (at least in theory).

A simpler problem

Still Boolean variables, but simpler constraints, only

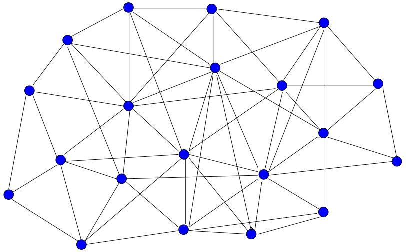
$x_i \neq x_j$

for some pairs (i, j).

I.e. we demand that some pairs of variables are not equal.

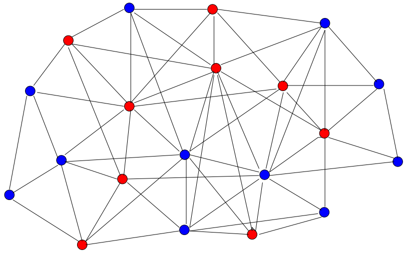
A graph problem

Max-cut. Given a graph divide the vertices into two sets to cut as many edges as possible.



A graph problem

Max-cut. Given a graph divide the vertices into two sets to cut as many edges as possible.



Indeed simple

It is easy to check whether we can cut all edges.

How about approximation?

Suppose we can cut 99% of the edges, can we efficiently find such a cut?

A brilliant idea

Goemans and Williamson: A cut is described by $x_i \in \{-1, 1\}$ and has value

$$\sum_{(i,j)\in E} \frac{1 - x_i x_j}{2}.$$

A brilliant idea

Goemans and Williamson: A cut is described by $x_i \in \{-1, 1\}$ and has value

$$\sum_{(i,j)\in E} \frac{1 - x_i x_j}{2}.$$

Introduce new variables y_{ij} and maximize

$$\sum_{(i,j)\in E} \frac{1 - y_{ij}}{2}$$

.

A brilliant idea

Goemans and Williamson: A cut is described by $x_i \in \{-1, 1\}$ and has value

$$\sum_{(i,j)\in E} \frac{1 - x_i x_j}{2}.$$

Introduce new variables y_{ij} and maximize

$$\sum_{(i,j)\in E} \frac{1 - y_{ij}}{2}$$

.

and requiring that the y-variables form a positive semi-definite matrix with ones on the diagonal.

Positive semidefinite matrices?

Y symmetric matrix is positive semidefinite iff one of the following is true

- All eigenvalues $\lambda_i \geq 0$.
- $z^T Y z \geq 0$ for any vector $z \in R^n$.
- $Y = V^T V$ for some matrix V.

$y_{ij} = x_i x_j$ is in matrix language $Y = xx^T$.

SDP is efficient

We can to any desired accuracy solve

$$\max \sum_{ij} c_{ij} y_{ij}$$

subject to

$$\sum_{ij} a_{ij}^k y_{ij} \le b^k$$

and Y positive semidefinite.

Intuitive reason, the set of PSD matrices is convex and we should be able to find optimum of linear function as we have no local optima (as is true for LP).

View using $Y = V^T V$

Want to solve

$$\max_{x \in -1,1^n} \sum_{(i,j) \in E} \frac{1 - x_i x_j}{2}.$$

but as $Y = V^T V$ we instead maximize

$$\sum_{(i,j) \in E} \frac{1 - (v_i, v_j)}{2}.$$

for $\|v_i\| = 1$, i.e. optimizing over vectors instead of real numbers.

Going vector to Boolean

The vector problem accepts a more general set of solutions. Gives higher objective value.

Key question: How to use the vector solution to get back a Boolean solution that does almost as well.

Rounding vectors to Boolean values

Great suggestion by Goemans and Williamson.

Given vector solution v_i pick random vector r and set

$$x_i = \mathrm{Sign}((v_i, r)),$$

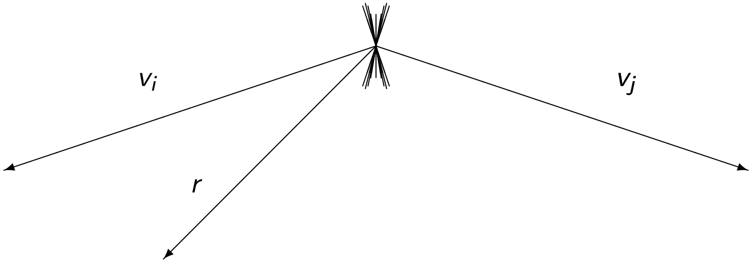
where (v_i, r) is the inner product.

Intuition of rounding

Contribution to objective function large,

$$\frac{1-(v_i, v_j)}{2}$$

large implying angle between v_i, v_j large,
$\mathrm{Sign}((v_i, r)) \neq \mathrm{Sign}((v_j, r))$ likely



Analyzing GW

Do term by term, θ angle between vectors.
Contribution to semi-definite objective function

$$\frac{1-(v_i,v_j)}{2} = \frac{1-\cos\theta}{2}$$

Probability of being cut

$$Pr[\mathrm{Sign}((v_i,r)) \neq \mathrm{Sign}((v_j,r))] = \frac{\theta}{\pi}$$

Minimal quotient gives approximation ratio

$$\alpha_{GW} = \min_{\theta} \frac{2\theta}{\pi(1-\cos\theta)} \approx .8785$$

Best possible?

Theorem: [H, 2001] Unless NP=P it is impossible to approximate Max-Cut within $16/17 \approx .941$.

Theorem: [KKMO,MOO, 2004]: If the Unique Games Conjecture (UGC) is true, then the Goemans-Williamson constant ($\approx .878$) is optimal.

One form of UGC

Variables x_i with values in $[m] = 0, 1 \ldots m-1$ and constraints:

$$\begin{aligned}
x_3 &= x_1 + 11 \bmod m \\
x_4 &= x_1 + 15 \bmod m \\
x_3 &= x_2 + 73 \bmod m \\
x_5 &= x_2 + 47 \bmod m \\
x_3 &= x_2 + 16 \bmod m \\
x_6 &= x_3 + 3 \bmod m \\
x_4 &= x_2 + 111 \bmod m
\end{aligned}$$

Linear equations modulo m with two variables in each equation.

"2-Lin-m"

An approximation problem

Distinguish 2-Lin-m instances where we can satisfy a fraction 99% of the equations from a instances where the best solution satisfies a fraction 1 %?

The Unique Games Conjecture (UGC) says that this is impossible to do efficiently for sufficiently large m.

Unique Games Conjecture

UGC has many consequences, not only that Goemans Williamson is optimal for Max-Cut.

Been around for over 10 years.

- It is difficult to find an algorithm that works for all instances.
- It is difficult to construct hard instances.

We seem to need something new to resolve it.

Again returning to TSP

TSP can be efficiently approximated within 1.5.

It is hard to approximate it within 123/122.

Some people are content "Within a constant, problem settled".

Some people are unhappy "We need to know this basic constant of nature".

Asymmetric TSP

Suppose that it might be longer (or more costly) to go from A to B than the other way around. “A road with hills”.

The approximation algorithm by Christofides breaks down.

State of the art, Asymmetric TSP

Best algorithm gives factor $c \log n / \log \log n$.

Best hardness result remains a small constant (75/74).

Historically algorithms have been found before hardness proofs.

Embarrassing fact

There is an algorithm based on Linear Programming that might give the answer to Asymmetric TSP within a factor of 2.

It was proposed by Held and Karp has been around for 50 years and we have been unable to analyze it.

Final words

- There are more or less tight approximation results for many problems. Max-3-Sat is just one example.
- Simple to state problems like ATSP, TSP, Max-Cut are not fully understood.
- Unique Games Conjecture resolve some open questions but not all.

The end

THE END